Computers and Geotechnics 171 (2024) 106374

Contents lists available at ScienceDirect

Computers and Geotechnics

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

Research paper

Inverse analysis of granular flows using differentiable graph neural network simulator

Yongjin Choi *, Krishna Kumar

Maseeh Department of Civil Architectural and Environmental Engineering Austin, The University of Texas at Austin, Austin, 78712, TX, USA

ARTICLE INFO

Keywords:
Inverse analysis
Granular flows
Differentiable simulator
Graph neural networks
Gradient-based optimization
Automatic differentiation

ABSTRACT

Inverse problems in granular flows, such as landslides and debris flows, involve estimating material parameters or boundary conditions based on a target runout profile. Traditional high-fidelity simulators for these inverse problems are computationally expensive, restricting the number of possible simulations. These simulators are also non-differentiable, making efficient gradient-based optimization methods for high-dimensional spaces inapplicable. Machine learning-based surrogate models offer computational efficiency and differentiability. However, they often struggle to generalize beyond their training data due to relying on low-dimensional input–output mappings that fail to capture the complete physics of granular flows. We propose a novel differentiable graph neural network simulator (GNS) that combines reverse mode automatic differentiation of graph neural networks with gradient-based optimization for solving inverse problems. GNS learns the dynamics of granular flow by representing the system as a graph and predicts the evolution of the graph at the next timestep, given the current state. The differentiable GNS demonstrates optimization capabilities beyond the training data. We demonstrate the effectiveness of our method for inverse estimation across single and multi-parameter optimization problems, including evaluating material properties and boundary conditions for a target runout distance and designing baffle locations to limit a landslide runout. Our proposed differentiable GNS framework solves inverse problems with orders of magnitude faster convergence than the conventional gradient-based optimization approach using finite difference.

1. Introduction

In geotechnical engineering, addressing optimization problems, such as the inverse modeling of landslides to infer material properties from target runout distances and design protective structures, is essential. Traditionally, this inverse optimization process has been approached by iteratively adjusting material properties to closely match observed results, often resulting in oversimplified, integer-valued material parameters. Design problems focus on optimizing geometries, such as the location and geometry of protective structures, to effectively mitigate and control granular flows (Liu et al., 2023; Babu and Basha, 2008). Addressing inverse and design optimization challenges is critical to developing robust models for managing landslide risks (Calvello et al., 2017; Cuomo et al., 2015; Abraham et al., 2021).

Inverse analysis often requires multiple forward simulations to adjust parameters until they match the desired outcome. Conventional high-fidelity forward simulators, such as discrete element method (DEM) (Staron and Hinch, 2005; Kermani et al., 2015; Kumar et al., 2017a) and Material Point Method (MPM) (Mast et al., 2014; Kumar et al., 2017b), are computationally intensive. This limits their practicality for repeated evaluations and constrains the range of parameters that can be effectively analyzed. The lack of comprehensive parametric sweeps results in unrealistic parameters that fit the target runout or sub-optimal designs. Although simplified depth-averaged models are computationally efficient (Hungr and McDougall, 2009; Christen et al., 2010; Mergili et al., 2017), they often fail to capture the full complexity of granular flow dynamics and are still limited to a finite set of parametric simulations. This issue is compounded as the parameter space becomes more complex and multi-dimensional.

Sampling-based methods like grid search (Ensor and Glynn, 1997), cross-entropy (Ho and Yang, 2010), and Bayesian optimization (Frazier, 2018) often require a significant number of forward simulations, especially as the number of parameters increases. In contrast, gradient-based optimization utilizes derivatives of the objective function to navigate the parameter space efficiently. This approach often leads to faster convergence with fewer simulations but requires the gradient of the objective function. However, traditional simulators are not differentiable. While finite difference methods can approximate the derivatives,

* Corresponding author.
E-mail address: yj.choi@utexas.edu (Y. Choi).

https://doi.org/10.1016/j.compgeo.2024.106374
Received 17 January 2024; Received in revised form 19 April 2024; Accepted 20 April 2024
Available online 3 May 2024
0266-352X/© 2024 Elsevier Ltd. All rights reserved.

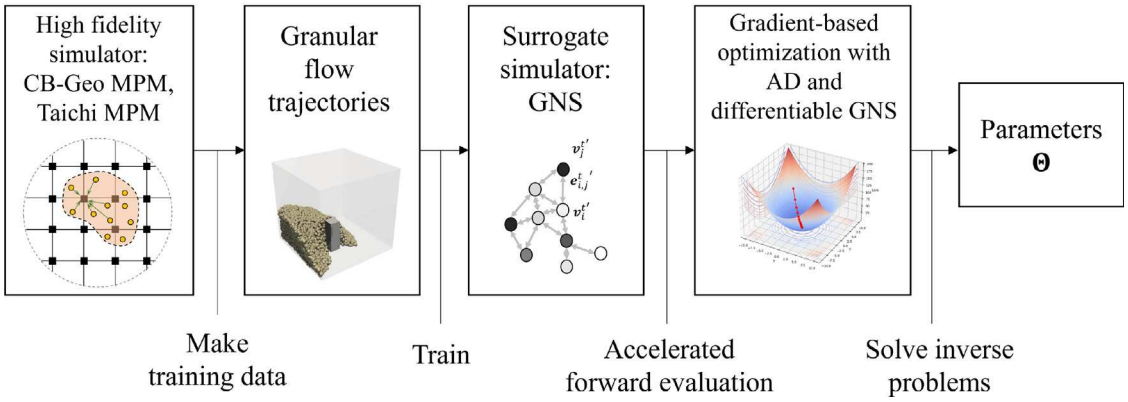

Fig. 1. Overview of the proposed workflow for inverse analysis of granular flows with differentiable graph network simulator.

they require a smaller increment size and are susceptible to numerical errors.

A popular gradient-based optimization approach is the adjoint method (Pires and Miranda, 2001; Cheylan et al., 2019). The adjoint method computes the gradients by introducing an auxiliary adjoint variable λ that satisfies an adjoint equation involving the derivatives of the forward problem. By solving this single adjoint equation, we can efficiently compute the gradients of the objective function f for all variables simultaneously without having to compute the derivatives of f numerically. However, the complexity and mathematical challenges inherent in solving adjoint equations, particularly for non-linear equations or problems involving discontinuities like friction, often necessitate alternative approaches.

In this context, Machine Learning (ML)-based surrogate models emerge as a compelling alternative for their efficiency and differentiability. These models create a non-linear functional mapping between influence factors – such as geometry, boundary conditions, and material properties – and the outcomes of granular flows, like runouts (Zeng et al., 2021; Ju et al., 2022). Despite their efficiency, the key limitation of ML-based models lies in their low-dimensionality mapping between a finite set of input parameters and output response, which does not fully encapsulate the underlying physics that governs flow behavior. For example, regression-based ML models could learn the relation between the material property and runout of granular columns but fail to generalize to other geometries or material properties. Consequently, these models often face challenges in generalizing beyond their initial training datasets, indicating a need for surrogate models that capture the complex granular flow behavior.

Recent advancements in learned physics simulators offer a promising solution to these limitations (Battaglia et al., 2016, 2018; Sanchez-Gonzalez et al., 2020). Graph neural network (GNN)-based simulators (GNSs) are one such learned physics simulator that represents the domain as a graph with vertices and edges and learns the local interactions between the vertices, which are critical in governing the dynamics of physical systems. By learning the interaction law, GNS demonstrates high predictive accuracy and generalization ability in modeling granular flows beyond the training data (Choi and Kumar, 2023a). Since GNS is based on neural network foundations, it is inherently differentiable through automatic differentiation (AD), making it suitable for gradient-based optimization (Allen et al., 2022; Zhao et al., 2022).

Automatic Differentiation (AD) (Baydin et al., 2018) uniquely bridges the gap between the computational rigor required for differentiating complex functions and the practical necessity of efficiency in neural network-based systems like GNS. In contrast to traditional differentiation methods such as manual, numerical, or symbolic differentiation, AD systematically breaks down functions into simple differentiable operations. This process enables precise derivative calculations without the approximation errors inherent in numerical methods or the complexity escalation typical of symbolic differentiation. AD-based simulators are increasingly popular in solving inverse problems in fluid dynamics (Wang and Kumar, 2023) and optimizing control for robotics (Hu et al., 2019).

In the context of GNS, we leverage reverse-mode AD to compute the gradients required for solving the inverse problem. Reverse-mode AD, commonly known as backpropagation (Hecht-Nielsen, 1992; LeCun et al., 1988) in neural networks, calculates gradients by first performing a forward pass through the network, storing intermediate values. It then executes a backward pass, efficiently propagating gradients from the output back to the inputs. This approach is particularly suited for GNS, where the number of parameters (inputs) significantly exceeds the number of outputs (e.g., a loss function). Using the AD gradient, we optimize the input parameters through gradient-based methods. By leveraging the computational efficiency and differentiability of GNS through AD, our method offers an efficient framework for solving inverse and design problems in granular flows.

A key challenge in applying AD to GNS, especially in simulations with extended temporal scales, is the extensive GPU memory requirement (Zhao et al., 2022). AD necessitates tracking all intermediate computations of GNS over the entire simulation for gradient evaluation. To mitigate this issue, we employ gradient checkpointing proposed by Chen et al. (2016). This approach significantly reduces the memory demand by strategically storing only key intermediate computational steps during gradient evaluation. While this technique conserves memory, it requires re-evaluating certain forward computations during the backward pass for gradient evaluation. This hybrid approach allows for monitoring gradients across thousands of particle trajectories over prolonged simulation durations without overwhelming GPU memory, albeit with an increased computational burden.

We introduce a novel framework for inverse analysis in granular flows, leveraging the automatic differentiation capabilities of GNS with gradient-based optimization. We demonstrate our gradient-based optimization framework on inverse problems and design applications in granular flows. The overview of the research is shown in Fig. 1. We first create granular flow trajectories using the material point method (MPM) (Soga et al., 2016; Kumar et al., 2019; Hu et al., 2018). The trajectories are stored as NPZ files (compressed NumPy file format) and are used to train GNS. We then use GNS as a surrogate to accelerate the forward evaluation of granular flow runouts. To solve the inverse problem, we leverage the reverse-mode Automatic Differentiation of GNS (AD-GNS) nature of GNS to calculate the derivatives of the target with respect to the input parameters. Finally, we use the gradients to optimize the parameters of interest based on a target iteratively. In the following sections, we describe the details of the proposed differentiable AD-GNS and gradient-based optimization approaches for solving inverse problems.

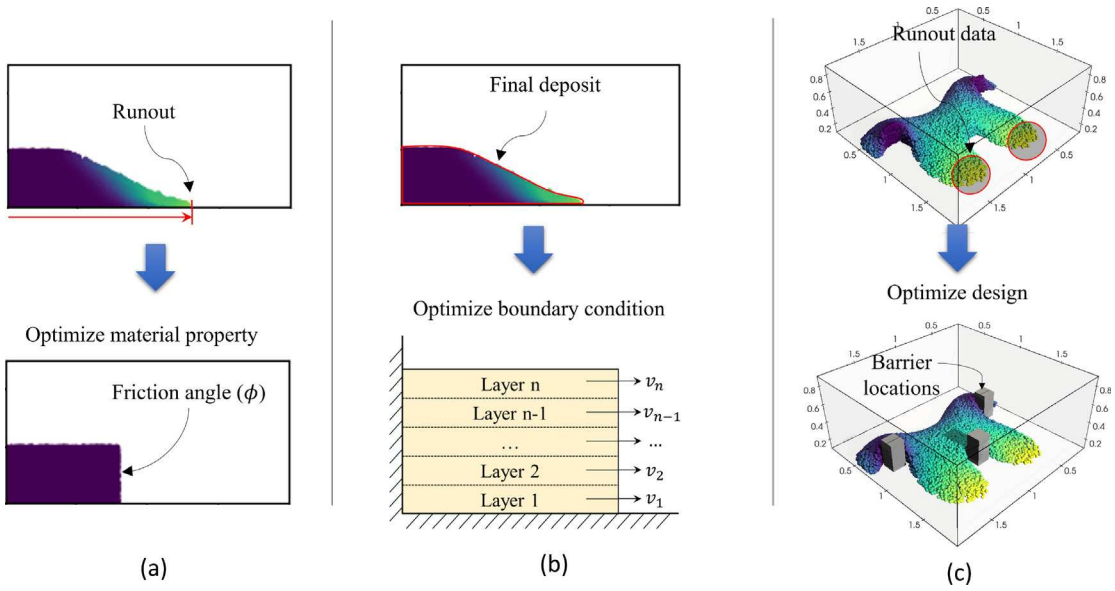

Fig. 2. Illustrative representation of inverse problems in granular media: (a) single parameter inverse of determining material property based on the final runout, (b) evaluating the initial boundary conditions (initial velocity) based on the final runout profile, and (c) optimizing the location of barriers based on the target runout distribution.

2. Problem statement

Inverse problems involve determining the underlying parameters from observed outcomes. We investigate three inverse problems in granular flows using the differentiable GNS framework, as shown in Fig. 2. We evaluate (a) single parameter inverse of determining material property based on the final runout, (b) multi-parameter inverse of evaluating the initial boundary conditions, i.e., identify the initial velocities of the n-layered granular mass given the final runout profile, and (c) optimizing the location of barriers based on the target runout distribution, such as the centroid of the final deposit. In these problems, we optimize by calculating the gradient of the input parameter(s) with respect to the observed output target.

Fig. 3 illustrates our framework to solve inverse problems in granular flows using differentiable GNS with AD (AD-GNS). Based on an initial set of input properties, we use the GNS to solve the forward problem of granular flow (Fig. 3a). Given a target parameter, we evaluate a loss function ($J_{\boldsymbol{\Theta}}$), e.g., the error between the observed runout ($R^{\boldsymbol{\Theta}}$) and the target runout ($R^{\boldsymbol{\Theta}_{target}}$). Using reverse-mode AD, we compute the gradient of the loss function with respect to the optimization parameter $\nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}}$. We then iteratively update the optimization parameter ($\boldsymbol{\Theta}$) using a gradient-based optimization technique with a learning rate η (Fig. 3b). This process continues until the loss decreases below a threshold, thus achieving the target response. We demonstrate GNS's ability to solve inverse problems by optimizing material properties, initial boundary conditions, and design of barriers (Fig. 2). In the following sections, we introduce GNS and gradient-based optimization.

3. Methods

3.1. Graph neural network-based forward simulator (GNS)

GNS is an efficient surrogate for high-fidelity simulations of granular media. GNS discretizes the domain into a set of vertices V on a graph (G), with each vertex representing an individual particle or a region and its properties, and the edges E represent the interaction between these regions. GNS takes the current state of the domain $\boldsymbol{X}_t$ at time t and returns its next state $\boldsymbol{X}_{t+1}$ (i.e., $GNS : \boldsymbol{X}_t \rightarrow \boldsymbol{X}_{t+1}$). $\boldsymbol{X}_t$ contains information about particles' position, velocity, distance to boundaries, and material properties. A surrogate simulation of granular flow involves running GNS through k timesteps predicting from the initial state $\boldsymbol{X}_0$ to $\boldsymbol{X}_k$ (i.e., $\boldsymbol{X}_0 \rightarrow \boldsymbol{X}_1 \rightarrow \cdots \rightarrow \boldsymbol{X}_k$) sequentially. We call this successive forward GNS prediction the "rollout". In the following paragraphs, we briefly introduce the structure of GNS. For more details about GNS, please see Sanchez-Gonzalez et al. (2020) and Choi and Kumar (2023a).

GNS consists of two components (Fig. 4): dynamics approximator D_θ, which is a learned function approximator parameterized by θ, and update function $\mathcal{U}$. D_θ has an encoder-processor-decoder architecture. The encoder converts $\boldsymbol{X}_t$ to a latent graph $G = (\boldsymbol{V}, \boldsymbol{E})$ representing the state of particle interactions. The processor propagates information between vertices by passing messages along edges and returns an updated graph $G' = (\boldsymbol{V}', \boldsymbol{E}')$. In the physics simulations, this operation corresponds to energy or momentum exchange between particles. The decoder extracts dynamics $\boldsymbol{Y}_t$ of particles from the updated graph. The update function uses the dynamics to update the current state to the next state of material points ($\boldsymbol{X}_{t+1} = \mathcal{U}(\boldsymbol{X}_t, \boldsymbol{Y}_t)$). The update function $\mathcal{U}$ is analogous to the explicit Euler integration in numerical solvers. Hence, in our GNS, $\boldsymbol{Y}_t$ corresponds to the second derivative of the current state, i.e., acceleration.

The encoder and decoder operations employ multi-layered perceptrons (MLPs), while the processor comprises GNN, which contains the learnable parameters θ. We train this θ to minimize the mean squared error between the ground truth accelerations of the particles $\boldsymbol{A}_t$ and the predicted dynamics $\boldsymbol{Y}_t$ given the current state of particles $\boldsymbol{X}_t$ as the input. In our study, we use the material point method (MPM) (Soga et al., 2016; Kumar et al., 2019) to generate the training dataset of particles (material point) positions $\boldsymbol{X}_t$ and acceleration $\boldsymbol{A}_t$.

We leverage PyTorch's Distributed Data Parallelism (DDP) framework to enable multiple GPU parallelization. This approach demonstrates almost linear scaling performance (Kumar and Vantassel, 2022; Choi and Kumar, 2023b). By employing DDP, we can distribute the workload associated with large graph structures across multiple GPUs with batching, thus facilitating the operation of GNS with a distributed computational burden without compromising the speed.

3.1.1. Training data

We prepare two types of datasets: two-dimensional granular flows (Flow2D) and three-dimensional granular flows interacting with obstacles (Obstacle3D). Table 1 shows the details of the simulation configurations.

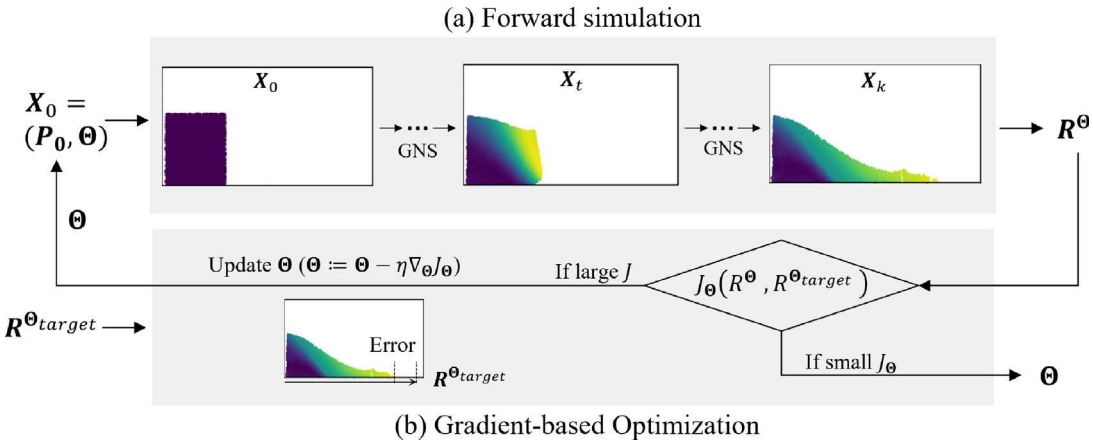

Fig. 3. Schematic diagram for differentiable GNS with AD (AD-GNS) for inverse analysis. It estimates the parameters Θ in granular column flow given a target runout distance $R^{\Theta_{target}}$.

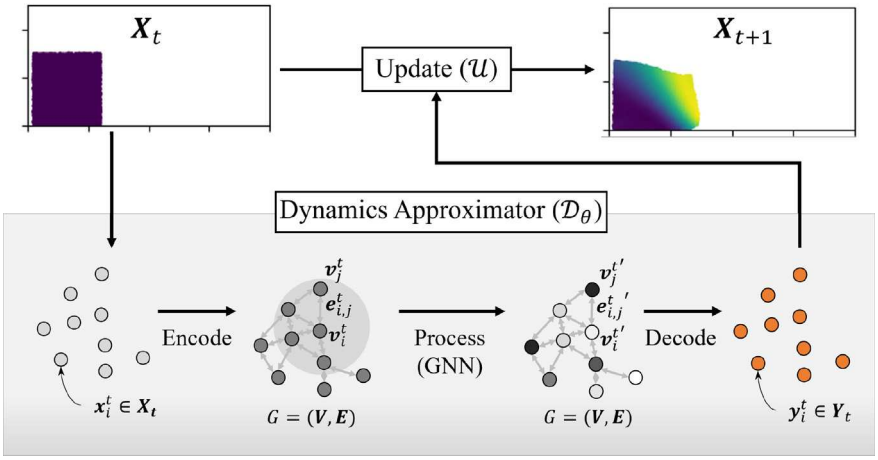

Fig. 4. Components of graph neural network (GNN)-based learned simulator (GNS).

We train the GNS model for the two-dimensional domain (Fig. 2a and b) on the Flow2D dataset. It includes 385 square-shaped granular mass flow trajectories in a two-dimensional box boundary. The appendix section provides a visual example in Fig. B.1. We use five different friction angles ($\phi = 15°, 22.5°, 30°, 37.5°, 45°$), where each friction angle has 77 simulation datasets. Each simulation has different initial configurations regarding the size of the square granular mass, position, and velocity. We use the explicit time integration method using CB-Geo MPM code (Kumar et al., 2019) to generate the training data.

We train the 3D GNS (see Fig. 2c) on the Obstacle3D dataset. It includes 1000 trajectories of granular mass interacting with barriers in a three-dimensional box boundary. The appendix section provides a visual example in Fig. B.2. The initial geometry of the cuboid-shaped granular mass varies from 0.25 to 0.80 m and is excited with different initial velocities. For each simulation, the granular mass interacts with one to three cuboid-shaped barriers whose width and length vary from 0.10 to 0.13 m with a height of 0.3 m in the $1.0 \times 1.0 \times 1.0$ m domain. We use the explicit time integration method using Moving Least Squares (MLS)-MPM code (Taichi MPM) (Hu et al., 2018) to generate the training data. See Appendix A for the forward simulation performance of GNS.

We use two different MPM software, CB-Geo MPM (2D simulations) and Taichi MPM (3D simulations), to get the best of both worlds. While CB-Geo MPM can reasonably simulate 2D granular flow, a 3D MPM code with rigid body dynamics was needed to model the barrier problems—hence, we use Taichi MPM code.

3.2. Differentiable GNS

GNS built using neural networks is fully differentiable (Allen et al., 2022) using reverse-mode AD. Reverse-mode AD computes the gradient of the output of GNS with respect to its inputs by constructing a computational graph of the simulation and propagating a gradient through the graph. Using the gradient, we can solve inverse problems using gradient-based optimization (Fig. 3b). This gradient-based optimization allows very efficient parameter updates, particularly in high-dimensional space, as it uses the gradient information to move the optimization parameters Θ towards the optimum solution (Allen et al., 2022; Dhara and Sen, 2023). Here, we describe the details of reverse-mode AD, followed by the gradient-based optimization in the later section (Section 3.3).

Reverse-mode AD (Baydin et al., 2018) offers a highly efficient method for calculating the accurate gradients of functions. Unlike numerical differentiation, which is approximate and computationally expensive, or symbolic differentiation, which can be unwieldy for complex functions, reverse-mode AD efficiently computes analytical gradients.

Reverse-mode AD leverages computational graphs (Fig. 5) to represent complex functions and efficiently and accurately compute their derivatives. A computational graph is directed where each node represents an elementary operation or variable, and edges indicate the computation flow. The combination of the nodes and edges makes computational graphs represent complex functions.

Table 1
Details of the Material Point Method (MPM) simulation geometries and properties used for generating the training datasets. *Source:* Reproduced after Choi and Kumar (2023a).

Property		Datasets	
		Flow2D	Obstacle3D
Simulation boundary		1.0 × 1.0 m	1.0 × 1.0 × 1.0 m
MPM element length		0.01 × 0.01 m	0.03125 × 0.03125 × 0.03125 m
Material point configuration		40,000 points/m^2	262,144 points/m^3
Granular mass geometry		0.2 × 0.2 to 0.4 × 0.4 m	0.25 to 0.80 m for each dimension
Max. number of particles		6.4K	17K
Barrier geometry		None	0.10 × 0.3 × 0.10 to 0.13 × 0.3 × 0.13 m
Simulation duration (# of timesteps)		1.0 s (400)	0.875 s (350)
Material property	Model	Mohr–Coulomb	Mohr–Coulomb
	Density	1800 kg/m^3	1800 kg/m^3
	Young's modulus	2 MPa	2 MPa
	Poisson ratio	0.3	0.3
	Friction angle	15, 22.5, 30, 37.5, 45 °	35°
	Cohesion	0.1 kPa	None
	Tension cutoff	0.05 kPa	None

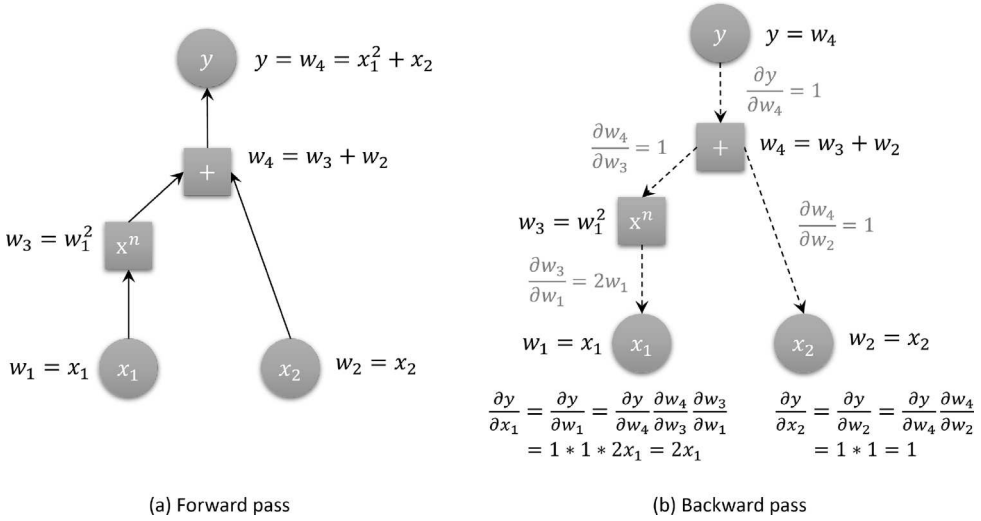

Fig. 5. Schematics of a Directed Acyclic Graph (DAG) of reverse-mode automatic differentiation (AD) of an example function.

To compute the gradient, reverse-mode AD conducts two steps: forward and backward pass. In the forward pass (Fig. 5a), reverse-mode AD evaluates the function's output given the inputs. Starting from the input nodes, which represent the input variables (x_1 and x_2), the graph is traversed to compute the output of the function. This step lays the groundwork for the differentiation process by establishing the relationships between all the operations involved in computing the function.

The next step is backward pass (Fig. 5b), known as reverse mode or backpropagation, which is the core process of reverse-mode AD to compute the gradient with respect to the inputs of the function (x_1 and x_2). This phase starts from the output of the computational graph (y) and works backward. As the process moves backward through the graph, reverse-mode AD computes what is known as a local derivative—the derivative of that node's operation with respect to its immediate input. In Fig. 5b, this corresponds to $\partial y/\partial w_4$, $\partial w_4/\partial w_3$, $\partial w_3/\partial w_1$, and $\partial w_4/\partial w_2$, denoted beside the gray dashed edges. By representing the simulation as a computational graph, we can pre-compute the derivatives, thus improving computational efficiency in gradient evaluation. We employ the chain rule, which combines every local derivative at each node from the output to the input node, returning the gradient of output with respect to the graph's input ($\partial y/\partial x_1$, $\partial y/\partial x_2$). In Fig. 5b, this process is described below the nodes x_1 and x_2.

The forward pass of the GNS is described as:

$$\boldsymbol{X}_{t+1} = GNS(\boldsymbol{X}_t) \quad \text{for } t \in \{0, 1, \ldots, k-1\} \tag{1}$$

where $\boldsymbol{X}_t$ is the current state of the system at time t, and GNS is the learned function approximator that predicts the next state $\boldsymbol{X}_{t+1}$ given the current state $\boldsymbol{X}_t$. The forward pass involves predicting the trajectory for k timesteps to simulate the evolution of the system from the initial state $\boldsymbol{X}_0$ to $\boldsymbol{X}_k$.

For the inverse analysis, we define an objective function (loss) $J_{\boldsymbol{\Theta}}$ that needs to be minimized with respect to the parameters $\boldsymbol{\Theta}$. The objective function depends on the output of GNS after rolling out for k timesteps, starting from the initial state $\boldsymbol{X}_0$.

$$J_{\boldsymbol{\Theta}} = J(\boldsymbol{X}_k) \tag{2}$$

where $\boldsymbol{X}_k$ is obtained by recursively applying $GNS()$ for k timesteps:

$$\boldsymbol{X}_1 = GNS(\boldsymbol{X}_0; \boldsymbol{\Theta})$$
$$\boldsymbol{X}_2 = GNS(\boldsymbol{X}_1; \boldsymbol{\Theta})$$
$$\vdots$$
$$\boldsymbol{X}_k = GNS(\boldsymbol{X}_{k-1}; \boldsymbol{\Theta})$$

We can compactly represent this recursive forward pass as:

$$\boldsymbol{X}_k = GNS^k(\boldsymbol{X}_0; \boldsymbol{\Theta}) \tag{3}$$

where GNS^k denotes applying $GNS()$ recursively for k timesteps.

To compute the gradient of $J_{\boldsymbol{\Theta}}$ with respect to $\boldsymbol{\Theta}$ using reverse mode AD, we construct the computational graph for the entire forward pass GNS^k and then apply the chain rule:

$$\nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}} = \frac{\partial J}{\partial \boldsymbol{X}_k}\frac{\partial \boldsymbol{X}_k}{\partial \boldsymbol{\Theta}} \tag{4}$$

where $\frac{\partial \boldsymbol{X}_k}{\partial \boldsymbol{\Theta}}$ is computed by recursively applying the chain rule through the entire trajectory:

$$\frac{\partial \boldsymbol{X}_k}{\partial \boldsymbol{\Theta}} = \frac{\partial \boldsymbol{X}_k}{\partial \boldsymbol{X}_{k-1}} \cdots \frac{\partial \boldsymbol{X}_1}{\partial \boldsymbol{X}_0} \frac{\partial \boldsymbol{X}_0}{\partial \boldsymbol{\Theta}} \tag{5}$$

This reverse mode AD computes the exact gradients $\nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}}$ by propagating the gradients from the final state $\boldsymbol{X}_k$ back to the initial state $\boldsymbol{X}_0$ and then to the parameters $\boldsymbol{\Theta}$, while leveraging the intermediate states $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_{k-1}$ computed during the forward pass.

Note that storing all these intermediate states for long trajectories (large k) can be memory-intensive, which is where gradient checkpointing (Section 3.3.1) is employed to reduce the memory footprint by selectively storing and recomputing intermediate states.

Unlike the finite difference method, which needs multiple function evaluations to estimate the derivative of each parameter, reverse-mode AD computes precise analytic gradients in a single forward pass. This makes reverse-mode AD particularly advantageous in gradient-based optimization, especially in large parameter space scenarios. The parameters can be efficiently updated towards the optimum based on the gradient information obtained from reverse-mode AD. In the following section, we elaborate on the gradient-based optimization with reverse-mode AD.

3.3. Gradient-based optimization

Eq. (6) shows the general concept of a simple gradient descent optimization scheme. Gradient descent minimizes the discrepancy between the simulator output and the target data, defined as an objective function $J_{\boldsymbol{\Theta}}$, by iteratively updating the model's parameter set $\boldsymbol{\Theta}$. The gradient of the objective function provides the information about the most effective update direction with respect to the parameters, $\nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}}$. Here, η is the learning rate.

$$\boldsymbol{\Theta} := \boldsymbol{\Theta} - \eta \cdot \nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}} \tag{6}$$

The learning rate or step size in an optimization algorithm determines the increment at each iteration while traversing towards a minimum of a loss function; this learning rate or step size η, multiplied by the gradient $\nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}}$, controls the aggressiveness of parameter updates in a gradient descent optimization of multi-variable function $J_{\boldsymbol{\Theta}}$ (see Eq. (6)). A high learning rate can lead to instability in the optimization process, with large steps causing overshooting of the minimum or even divergence. Conversely, a small learning rate will result in slow convergence, as the model takes minuscule steps and requires excessive iterations to reach a solution. Finding the optimal learning rate is largely experimental. We start with a small learning rate (e.g., 0.001), observe the parameter updates in the initial iterations, and manually vary the learning rate to achieve convergence in a reasonable time. We can also employ adaptive learning rates or learning rate schedulers that decrease the rate over time for stability. Alternatively, adaptive optimizers like adaptive movement estimation algorithm (ADAM) (Kingma and Ba, 2014) or root mean squared propagation (RMSprop) automate some of the learning rate adjustment process.

The differentiable simulator computes the gradient $\nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}}$ using reverse-mode AD to update the parameter set $\boldsymbol{\Theta}$ at each optimization step. Since GNS is a differentiable simulator, we can seamlessly utilize the AD tools provided by frameworks such as PyTorch to calculate the gradient efficiently.

This approach offers several advantages. First, it provides a systematic way of handling complex, nonlinear inverse problems where analytical solutions are not feasible. Second, the efficiency of reverse-mode AD in differentiable simulators enables handling high-dimensional parameter spaces and complex relationships between parameters and outputs. Finally, gradient-based optimization often leads to faster convergence to a solution compared to other sampling-based optimization methods, especially in large-scale problems (Allen et al., 2022).

We use the adaptive movement estimation algorithm (ADAM) (Kingma and Ba, 2014) for high-dimensional parameter optimization. ADAM incorporates a dynamic adaptive learning rate for each parameter using the moments of the gradients. ADAM accelerates the parameter updates in the right direction, thus avoiding local minima and adapting the learning rate for each parameter, making it less sensitive to the scale of the gradients.

Specifically, the parameters are updated as:

$$\boldsymbol{\Theta}_i = \boldsymbol{\Theta}_{i-1} - \frac{\eta}{\sqrt{\hat{v}_i} + \epsilon} \hat{m}_i \tag{7}$$

where $\boldsymbol{\Theta}$ is the parameter set with the optimization step i, and ϵ is a small scalar for numerical stability. The first and second moments (m_i and v_i) of the gradients are updated as:

$$m_i = \beta_1 m_{i-1} + (1 - \beta_1) g_i \tag{8}$$

$$v_i = \beta_2 v_{i-1} + (1 - \beta_2) g_i^2 \tag{9}$$

where g_i represents the gradient at step i, β_1 and β_2 are decay rates that control the exponential decay of these moving averages. Early in the training process, the variables m_i and v_i are initially biased towards zero because they start from zero. To address this bias, a correction is applied as:

$$\hat{m}_i = \frac{m_i}{1 - \beta_1^i} \tag{10}$$

$$\hat{v}_i = \frac{v_i}{1 - \beta_2^i} \tag{11}$$

This adaptive learning rate is particularly effective optimization in high-dimensional parameter spaces, which entails sparse and varied gradient scales. ADAM adapts the learning rate for each parameter based on its historical gradients, which helps in scenarios where different parameters have different scales or degrees of sparsity. This adaptive learning makes ADAM converge quicker and improves performance in solving multi-parameter inverse problems. We refer readers to Kingma and Ba (2014) for more technical details.

3.3.1. Gradient checkpointing

Reverse-mode AD requires significant memory for large-scale neural networks (Zhao et al., 2022). AD necessitates storing intermediate variables in the computational graph during the forward pass. These intermediate variables should be retained during the backward pass to apply the chain rule in computing gradients. The computational graph grows for large-scale neural networks by adding more layers and parameters. Therefore, the backpropagation requires substantial memory to retain all the intermediate variables in the increasingly large computational graph.

Since GNS contains multiple MLP and GNN layers containing millions of parameters, and the entire simulation even entails the accumulation of GNS ($\boldsymbol{X}_t \rightarrow \boldsymbol{X}_{t+1}$) for k steps, computing $\nabla_{\boldsymbol{\Theta}} J_{\boldsymbol{\Theta}}$ using reverse-mode AD requires extensive memory capacity. We found that conducting reverse-mode AD for entire simulation timesteps, which includes more than hundreds of steps, is not feasible in the currently available GPU memory capacity (40 GB). The forward pass fails in 3 to 4 timesteps due to the lack of memory when simulating granular flows with about 3K particles. To overcome this limitation, we employ gradient checkpointing.

Gradient checkpointing (Chen et al., 2016) is a technique to mitigate the prohibitive memory demands of storing all intermediate states within a computational graph during reverse-mode AD. In the specific setting of optimizing parameters with GNS, we face the challenge of an extensive computational graph that spans both GNN message-passing layers and simulation timesteps. Checkpointing addresses this by strategically saving a subset of the intermediate states (checkpoints) and discarding the rest. During the backward pass, any required states that are not checkpoints are recomputed on the fly from the most

recent checkpoint. This selective storage significantly reduces the peak memory requirement at the expense of some recomputation.

Let us consider the forward pass of GNS over k timesteps:

$$\boldsymbol{X}_0 \rightarrow \boldsymbol{X}_1 \rightarrow \boldsymbol{X}_2 \cdots \rightarrow \boldsymbol{X}_k \tag{12}$$

During the forward pass, instead of storing all the intermediate states $\boldsymbol{X}_0, \boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_{k-1}$, gradient checkpointing selectively stores only a subset of these states, called checkpoints, at specific timesteps $C \in \{0, c_1, c_2, \ldots, c_m, k\}$ as:

$$\boldsymbol{X}_i = GNS(\boldsymbol{X}_{i-1}); \quad \begin{cases} \text{store} \leftarrow \boldsymbol{X}_i & \forall i \in C \\ \text{discard} \leftarrow \boldsymbol{X}_i & \forall i \notin C \end{cases} \tag{13}$$

During the backward pass, those intermediate states between $\boldsymbol{X}_{i-1}$ and $\boldsymbol{X}_i \; \forall i \notin C$ are recomputed to evaluate the local derivatives. The reverse mode AD with gradient checkpoints is computed as follows:

1. **Initial gradient computation at final state:** Start by computing the gradient of the loss function J with respect to the final state $\boldsymbol{X}_k$: $\frac{\partial J}{\partial \boldsymbol{X}_k}$ This gradient is derived from the final output of the GNS model, based on how the loss J is affected by the last computed state $\boldsymbol{X}_k$.
2. **Backpropagate through GNS from $\boldsymbol{X}_k$ to last checkpoint $\boldsymbol{X}_{c_m}$:** Use the gradient from the final state to backpropagate through the operations performed from the last checkpoint to the final state:

$$\frac{\partial J}{\partial \boldsymbol{X}_{c_m}} = \left(\frac{\partial J}{\partial \boldsymbol{X}_k}\right) \cdot \left(\frac{\partial \boldsymbol{X}_k}{\partial \boldsymbol{X}_{c_m}}\right)$$

   Here, $\frac{\partial \boldsymbol{X}_k}{\partial \boldsymbol{X}_{c_m}}$ represents the gradient of the output at step k with respect to the state at the last checkpoint c_m, encompassing all intermediate transformations.
3. **Backpropagate successively through each checkpoint:** For each checkpoint c_m, starting from the last stored checkpoint c_m and moving backward through earlier checkpoints $c_{m-1}, c_{m-2}, \ldots$, compute:

$$\frac{\partial J}{\partial \boldsymbol{X}_{c_{(m-1)}}} = \left(\frac{\partial J}{\partial \boldsymbol{X}_{c_m}}\right) \cdot \left(\frac{\partial \boldsymbol{X}_{c_m}}{\partial \boldsymbol{X}_{c_{(m-1)}}}\right)$$

   Each term $\frac{\partial \boldsymbol{X}_{c_m}}{\partial \boldsymbol{X}_{c_{(m-1)}}}$ captures the gradient of the checkpointed state with respect to its preceding checkpoint, reflecting the operations of GNS executed between these points.
4. **Recompute non-stored states as needed:** Assuming a non-checkpointed intermediate state i between checkpoints $c_{(m-1)}$ and $c_{(m-2)}$, where $c_{(m-2)} < i < c_{(m-1)}$ and $\boldsymbol{X}_i$ is not stored during the forward pass. We handle the backward pass as:

   (a) **Recompute intermediate states:** If $\boldsymbol{X}_i$ is an intermediate state that was not stored, recomputation from the last stored state $\boldsymbol{X}_{c_{(m-2)}}$ is necessary:

$$\boldsymbol{X}_i = \text{GNS}^{i \leftarrow c_{(m-2)}}(\boldsymbol{X}_{c_{(m-2)}})$$

   This step involves executing the forward function GNS iteratively from $c_{(m-2)}$ to i, reconstructing each $\boldsymbol{X}_i$ sequentially up to $\boldsymbol{X}_{c_{(m-1)}}$.

   (b) **Backpropagate gradient through $\boldsymbol{X}_i$: With $\boldsymbol{X}_i$ recomputed, the gradient from $\boldsymbol{X}_{c_{(m-1)}}$ is propagated back:**

$$\frac{\partial J}{\partial \boldsymbol{X}_i} = \left(\frac{\partial J}{\partial \boldsymbol{X}_{c_{(m-1)}}}\right) \cdot \left(\frac{\partial \boldsymbol{X}_{c_{(m-1)}}}{\partial \boldsymbol{X}_i}\right)$$

   We compute the gradients for each $\boldsymbol{X}_j$ (for $j = k$ to $c_{(m-1)}$) during this backpropagation process.

   (c) **Gradient at checkpoint $c_{(m-2)}$:** We continue the gradient propagation back to $\boldsymbol{X}_{c_{(m-2)}}$:

$$\frac{\partial J}{\partial \boldsymbol{X}_{c_{(m-2)}}} = \left(\frac{\partial J}{\partial \boldsymbol{X}_i}\right) \cdot \left(\frac{\partial \boldsymbol{X}_i}{\partial \boldsymbol{X}_{c_{(m-2)}}}\right)$$

   This step involves recalculating each partial derivative $\partial \boldsymbol{X}_j / \partial \boldsymbol{X}_{j-1}$ during the recomputation phase.
5. **Finalize gradients for initial state and parameters:** Once backpropagation reaches the first checkpoint $\boldsymbol{X}_{c_1}$, complete the gradient computation for the initial state and model parameters:

$$\frac{\partial J}{\partial \boldsymbol{X}_0} = \left(\frac{\partial J}{\partial \boldsymbol{X}_{c_1}}\right) \cdot \left(\frac{\partial \boldsymbol{X}_{c_1}}{\partial \boldsymbol{X}_0}\right)$$

$$\frac{\partial J}{\partial \boldsymbol{\Theta}} = \left(\frac{\partial J}{\partial \boldsymbol{X}_{c_1}}\right) \cdot \left(\frac{\partial \boldsymbol{X}_{c_1}}{\partial \boldsymbol{\Theta}}\right)$$

   These calculations utilize the gradients at the first checkpoint to determine how initial conditions and parameters should be adjusted to minimize the loss J.

By selectively storing and recomputing intermediate states, gradient checkpointing reduces the peak memory requirement during backpropagation at the cost of some additional computations during the backward pass. Choosing checkpoint locations C is crucial for balancing memory savings and computational overhead. A common strategy is to place checkpoints at regular intervals (e.g., every few timesteps) or at specific points in the simulation where the computational graph is expected to be particularly large or complex. Using gradient checkpointing, we can differentiate through the entire simulation.

4. Results of inverse analysis in granular flows

We evaluate the performance of AD-GNS in solving the three inverse problems described in Fig. 2. Note that the GNS was not trained on any of these problems to showcase the adaptability and robustness of our approach.

4.1. Single parameter inverse

Estimating the material properties of granular media resulting in a landslide or debris flow is important for planning mitigation measures. In this problem, we aim to evaluate the friction angle ϕ of the granular column that produces a target runout distance $d_{\phi_{target}}$ (as outlined in Fig. 2a), where d_ϕ is the distance between the flow toe and the leftmost boundary. We use AD-GNS to generate a forward simulation to estimate d_ϕ for a given initial ϕ. We evaluate a loss J_ϕ as the squared error between $d_{\phi_{target}}$ and d_ϕ (Eq. (14)), then use AD-GNS to compute the gradient $\nabla_\phi J_\phi$. Using GD, we update the ϕ to find the target ϕ for a desired runout $d_{\phi_{target}}$ until the squared error becomes lower than 0.0005, the optimization threshold.

$$J_\phi = (d_\phi - d_{\phi_{target}})^2 \tag{14}$$

To test our method's versatility, we selected four granular column collapse scenarios with varying aspect ratios and friction angles not covered in our training data. These scenarios ranged from columns with a small aspect ratio (short column, $a = 0.5$) to those with a large aspect ratio (tall column, $a = 2.0$). Both cases have 3200 particles in the simulation.

Table 2 summarizes the scenarios and results for the inversely estimated friction angles. Fig. 6a shows the optimization history based on reverse-mode AD, and Fig. 8 shows the visual progress of the optimization for all the test scenarios. We set the learning rate η of 500 in Eq. (6), and the initial guess of the friction angle starts from 30°.

Table 2
Inverse analysis result for estimating friction angle ϕ in granular column collapse. $d_{\phi_{target}}$ is runout distance from MPM with ϕ_{target}. $d^{GNS}_{\phi_{target}}$ is runout distance from GNS with ϕ_{target}. d error is percentage error between $d_{\phi_{target}}$ and $d^{GNS}_{\phi_{target}}$. d^{GNS}_{ϕ} is runout distance from GNS with estimated ϕ. ϕ error is percentage error between estimated ϕ and ϕ_{target}. d error is percentage error between d^{GNS}_{ϕ} and $d_{\phi_{target}}$.

Scenario						Optimization				
Column type	$H_0 \times L_0$ (m)	ϕ_{target} (°)	$d_{\phi_{target}}$ (m)	$d^{GNS}_{\phi_{target}}$ (m)	d error (%)	Initial ϕ (°)	Estimated ϕ (°)	d^{GNS}_{ϕ} (m)	ϕ error (%)	d error (%)
Short	0.2 × 0.4	21	0.6947	0.7154	2.98	30	22.88	0.6953	8.94	0.08
		42	0.6000	0.6067	1.12	30	41.31	0.6209	1.65	3.49
Tall	0.4 × 0.2	21	0.8318	0.8783	5.58	30	22.87	0.8401	8.90	0.99
		42	0.5665	0.5942	4.89	30	44.54	0.5705	6.06	0.72

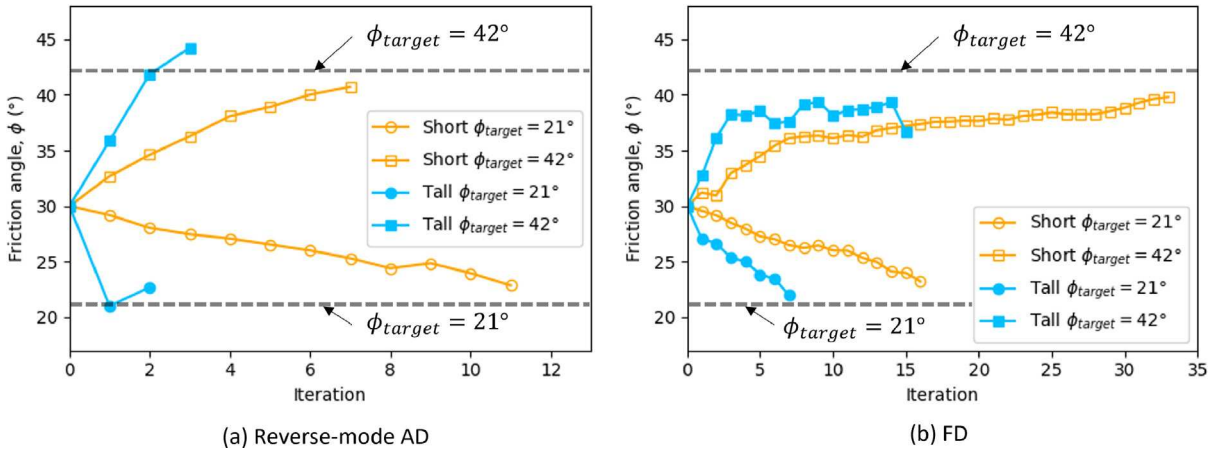

Fig. 6. Optimization history of friction angle with iterations from (a) reverse-mode AD and (b) finite differentiation (FD).

As can be seen in Fig. 6a, the optimization successfully converges to the friction angles ϕ that are close enough to the target values ϕ_{target} showing errors less than 8.94% (see Table 2) in a few iterations. We also compare the predicted runout distance d^{GNS}_{ϕ} with the inferred ϕ and target runout distance $d_{\phi_{target}}$. The predictions show accurate values with a maximum error of 3.49% (see Table 2). Although the inverse estimation of ϕ includes a small error (about 1° to 3°) compared to ϕ^{target}, the runout predictions are accurate within 3.49%. The major source of the error is the differences in runout distance computed by GNS and MPM. $d_{\phi_{target}}$ and $d^{GNS}_{\phi_{target}}$ represents the runout computed by MPM and GNS at ϕ_{target}, respectively. Although GNS and MPM use the same friction angle, the GNS includes a small amount of error (1.12% to 5.58%) since GNS is a learned surrogate for MPM.

Comparing our AD-based approach to the finite difference (FD) method reveals its stability. Fig. 6b shows the optimization history using FD where the gradient is estimated through two forward evaluations at ϕ and $\phi + \Delta\phi$. We use $\Delta\phi = 0.05$ and set the same learning rate η as AD. Despite using the same learning rate as AD, the FD method encounters challenges. The optimization is less stable, and for the tall column with $\phi_{target} = 42°$, it fails to identify the correct friction angle with the error of about 10%, which results in d error of 16%. This error stems from FD's reliance on the chosen $\Delta\phi$ for gradient approximation, unlike AD's precise gradient computations. Fig. 7 shows the loss history of reverse-mode AD and FD. We observe the loss trajectory of FD being trapped in the flat region between 36° to 39°. This is attributed to the diminutive update term $\eta \cdot \nabla_{\phi} J_{\phi}$, a consequence of underestimating gradients due to the selected $\Delta\phi$ value (= 0.05), as opposed to the accurate gradients provided by AD. The first-order forward FD approach has an error of $\mathcal{O}(\Delta\phi)$ in gradient computation.

In the visual progress of the optimization (see Fig. 8), the yellow dots represent the final deposit predicted by GNS, and the gray shade represents the final deposit corresponding to ϕ_{target} from MPM. For all scenarios, our approach effectively identifies ϕ that closely matches ϕ_{target}, as d_{ϕ} converges to $d_{\phi_{target}}$. Notably, the GNS not only accurately predicts the final runout distance but also captures the overall geometry of the granular deposit, although we only use d_{ϕ} for the optimization. Unlike typical low-dimensional empirical correlations, GNS offers prediction of full granular dynamics. By simulating the entire granular flow process, GNS enables our optimization to reproduce the runout distance and the detailed granular geometry of the deposit.

4.2. Multi-parameter inverse

Real-world inverse problems are complex as they include multiple parameters for optimization. In this section, we evaluate the performance of AD-GNS in solving the multi-parameter boundary condition inverse problem (as outlined in Fig. 2b). The objective is to determine the initial boundary condition, i.e., x-velocities ($\boldsymbol{v}$), of each layer in the multi-layered granular column that produces a target deposit $\boldsymbol{S}_{\boldsymbol{v}_{target}}$. Here, $\boldsymbol{S}_{\boldsymbol{v}}$ is the coordinate value of all material points when the flow stabilizes. Thus, $\boldsymbol{v}$ is our parameter set $\boldsymbol{\Theta}$, and $\boldsymbol{S}_{\boldsymbol{v}}$ is our runout metric to minimize with respect to $\boldsymbol{S}_{\boldsymbol{v}_{target}}$ with an initial $\boldsymbol{v}_{target}$ from MPM simulation.

Fig. 9 shows the granular column collapse scenario for the inverse analysis addressed in this section. The length and height of the column is 0.35 × 0.28 m with an aspect ratio of 0.8, including 3920 particles. It consists of 10 horizontal layers with the same thickness with linearly decreasing initial x-velocity from 1.50 m/s at the first layer to 0.15 m/s at the 10th layer., $\boldsymbol{v}_{target} = [1.50,\ 1.35,\ \ldots,\ 0.30,\ 0.15]$ m/s. The GNS has not encountered this discretized layered velocity condition during training.

Fig. 10 visualizes the GNS prediction of runout during optimization with different $\boldsymbol{v}$. We update the parameters by updating the ADAM with the learning rate $\eta = 0.1$. The loss $J_{\boldsymbol{v}}$ is defined as the mean squared error (MSE) of the coordinate values for materials points between the predicted final deposit $\boldsymbol{S}_{\boldsymbol{v}} = \{\boldsymbol{p}^i\}_{i=1:N}$ and the target final deposit $\boldsymbol{S}_{\boldsymbol{v}_{target}} = \{\boldsymbol{p}^i_{target}\}_{i=1:N}$ from MPM (Eq. (15)). Here, $\boldsymbol{p}^i_{\boldsymbol{v}}$ and $\boldsymbol{p}^i_{\boldsymbol{v}_{target}}$ denotes the coordinates of N materials points.

$$J_{\boldsymbol{v}} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{p}^i_{\boldsymbol{v}} - \boldsymbol{p}^i_{\boldsymbol{v}_{target}}\|^2 \tag{15}$$

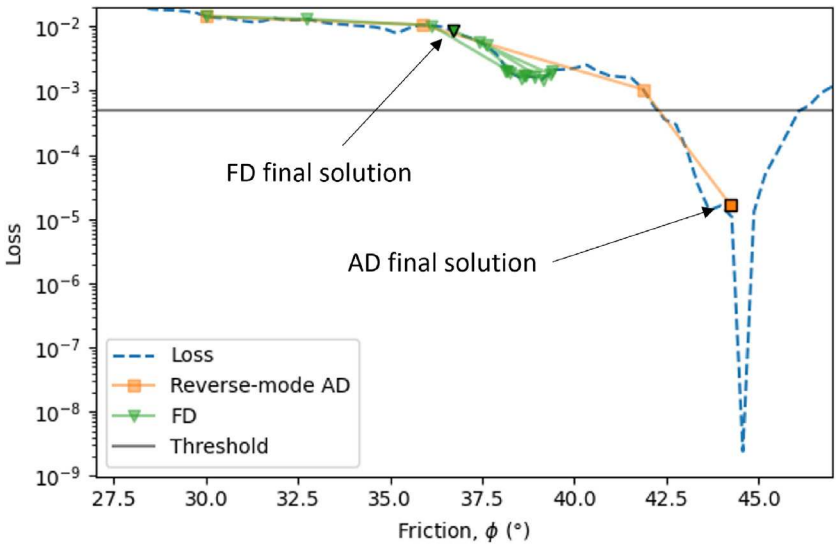

Fig. 7. Loss trajectory of reverse mode AD and FD.

The gray shade shows the final runout deposit produced from our scenario (Fig. 9) simulated using MPM, which is our target, and the yellow dots are the GNS prediction of the final deposit at the corresponding optimization iterations. The initial velocity guesses for each layer are set to zero, as shown by the cross (×) markers in Fig. 11a. As iterations progress, the velocities for each layer gradually align with $\boldsymbol{v}_{target}$, indicated by square markers in Fig. 11a, and consequently, the final deposit geometry closely matches the target at iteration 29 with MSE of 3.41e−4, as shown in Fig. 11b. We also test the GNS prediction with the target velocities (see Fig. 12). The GNS prediction with the target velocities (blue dashed line in Fig. 12) shows a good agreement with GNS prediction with optimized velocities at iteration 29 (yellow dashed line in Fig. 12) and the target (gray shade in Fig. 12).

Since GNS is a surrogate model of the forward simulator, it inherently incorporates errors; the inferred velocities might not reproduce the same final deposit geometry in the ground truth simulator, MPM. Hence, verifying if the inferred velocities are still valid in the ground truth simulator when reproducing the target final deposit is necessary.

To confirm the validity of the inferred velocities, we compare the final deposit with the inferred velocities using MPM with the target final deposit. Fig. 12 shows the MPM simulation results of the final deposit from the inferred velocity at the 29th iteration and target velocity. The gray shade represents the target deposit, and the red dots represent the deposit from MPM using the inferred velocities. The final deposit from the inferred velocity reasonably matches our target with a minor difference at the left boundary with an MSE of 8.70e−4, shown in Fig. 11b. The MSE is slightly larger than the optimization error at the 29th iteration but still significantly smaller than that of the initial iteration, which is about 0.01. The comparison suggests that our method provides reasonable velocity estimations to replicate the target behavior, even though GNS had not been exposed to the discretized layered initial velocity condition during training.

4.3. Design of baffles to resist debris flow

We can use AD-GNS to design engineering structures, which involves optimizing the design parameters of structural systems to achieve a specific functional outcome. We demonstrate the use of AD-GNS in the design of the debris-resisting baffles to achieve a target runout distance.

Debris-resisting baffles (Fig. 2c) are rigid flow-impeding earth structures strategically placed perpendicular to potential landslide paths to interrupt the flow of landslide debris (Yang and Hambleton, 2021). Their primary function is to reduce the risk and impact of debris flow by decelerating it and dissipating its energy upon impact. The configuration of these baffles is crucial as it directly influences their effectiveness in mitigating debris flow.

Our inverse analysis aims to optimally position the baffles to halt granular flow within a predefined area (as outlined in Fig. 2c) Specifically, we optimize the z-locations of two barriers, denoted as $\boldsymbol{z}$, to ensure that the toe of the granular jet—defined as the centroid $\boldsymbol{c}_z$ of the furthest 10% of downstream particles—stops at a specific target point $\boldsymbol{c}_{z_{target}}$. In this context, $\boldsymbol{z}$ corresponds to our parameter set $\boldsymbol{\Theta}$, and $\boldsymbol{c}_z$ represents our measure of runout $\boldsymbol{R}_{\boldsymbol{\Theta}}$ in Fig. 3.

The initial geometry of the granular mass (see Fig. 13) is cuboid-shaped with the size of 0.3 × 0.2 × 0.7 m for x, y, and z coordinates, and its lower edge is at (x = 0.2, y = 0.125, z = 0.25) m. The granular mass is subjected to a uniform initial x-velocity of 2.0 m/s. The size of the baffles is 0.15 × 0.2 × 0.15 m. The total number of particles is 13,371. The simulation domain is 2.0 × 1.0 × 1.0 m. The GNS has never seen this baffle size and the simulation domain during training, which is two times larger than the training domain (see Table 1 for comparison). These differences place our inverse solver in a more challenging circumstance.

Fig. 14 illustrates the visual progress of the optimization. We use ADAM for the optimization with the learning rate $\eta = 0.01$. The loss $J_{\boldsymbol{c}_z}$ is defined as the MSE between the coordinates $\boldsymbol{c}_z$ and $\boldsymbol{c}_{z_{target}}$, which is the absolute Euclidean distance:

$$J_{\boldsymbol{c}_z} = \|\boldsymbol{c}_z - \boldsymbol{c}_{z_{target}}\|^2 \tag{16}$$

The gray broken line shows the initial geometry of the granular mass, and the yellow dots represent the final deposit after flow ceases. An orange outline highlights the flow toe, with a purple dot marking its centroid $\boldsymbol{c}_z$. The black dot marks the target centroid location $\boldsymbol{c}_{z_{target}}$. We start the initial guess of the barrier center locations at (x = 0.675 m, z = 0.225 m) for the lower baffle and at (x = 0.675 m, z = 0.925 m) for the upper baffle, as illustrated by the gray boxes in Fig. 14. As iteration proceeds, the centroid of the flow toe converges to the target centroid with the baffles z-located at 0.341 m and 0.812 m. As depicted in Fig. 15, the loss history converges to a final loss of 4.68e−6 at iteration 23.

As we discussed in Section 4.2, we need to validate if the inferred baffle locations are still effective in the ground truth simulator in replicating the desired outcome. Fig. 16 shows the final deposit and the centroid based on the optimized baffle locations using MPM. Although there is a noticeable difference between the MPM centroid and the

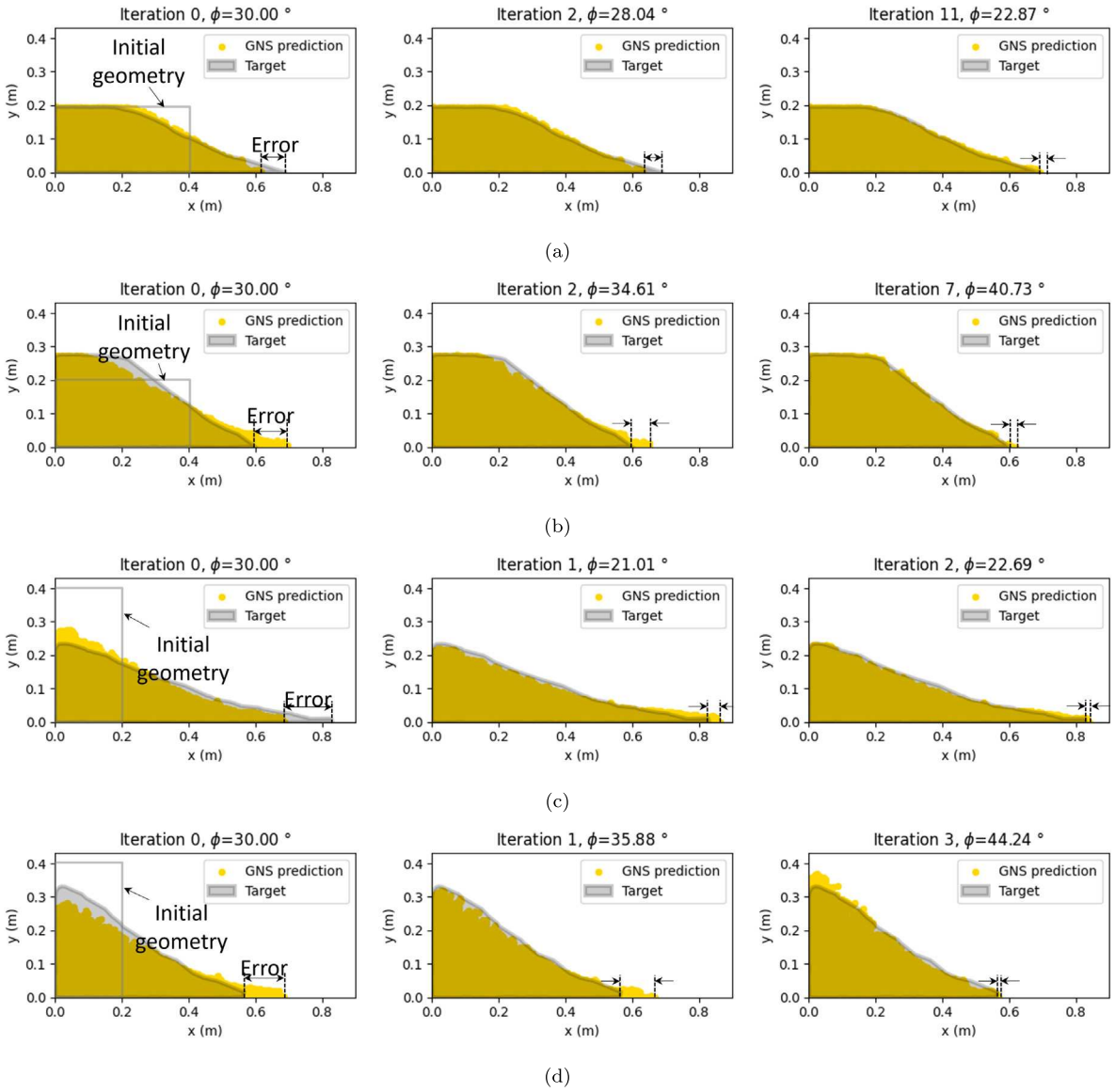

Fig. 8. Visualization of GNS prediction during optimization by varying ϕ: (a) short column ($a = 0.5$) with $\phi_{target} = 21°$, (b) short column ($a = 0.5$) with $\phi_{target} = 42°$, (c) tall column ($a = 2.0$) with $\phi_{target} = 21°$, (d) tall column ($a = 2.0$) with $\phi_{target} = 42°$. The yellow dots represent the final deposit predicted by GNS, and the gray shade represents the final deposit from MPM with to ϕ_{target}.

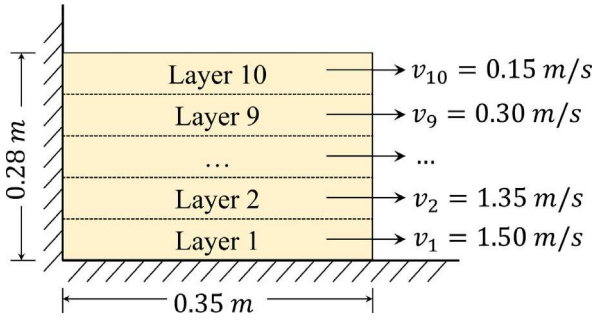

Fig. 9. Granular column collapse scenario.

target centroid, with the validation loss of 1.89e−3, this deviation is still markedly lower than the loss at the initial iteration (2.30e−2) in Fig. 14. This outcome is promising because we extrapolate beyond the training data.

An essential element in our inverse analysis framework is the implementation of gradient checkpointing on GNS rollout. This technique is highly advantageous, particularly when tracking gradients over long timesteps in 3D scenarios, such as the one described in this section. Without gradient checkpointing, we encountered significant limitations in forward steps due to GPU memory constraints, reaching the 40 GB limit within merely three steps of the rollout for the backpropagation. However, by employing gradient checkpointing and thus storing the sparse results of intermediate steps, we could track gradients effectively for hundreds of steps. This alleviated memory issues and ensured our approach's feasibility for larger and more complex scenarios.

4.4. Optimization efficiency

We compare the computation time between our proposed approach and the baseline Finite Difference methods (Table 3). The results are reported based on the single parameter optimization scenario about the tall column ($a = 2.0$) with $\phi = 42°$ shown in Fig. 8d. The baseline method involves executing two forward simulations ($\boldsymbol{X}_0, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_k$), where $k = 374$ employing our ground-truth simulator, MPM, at ϕ and

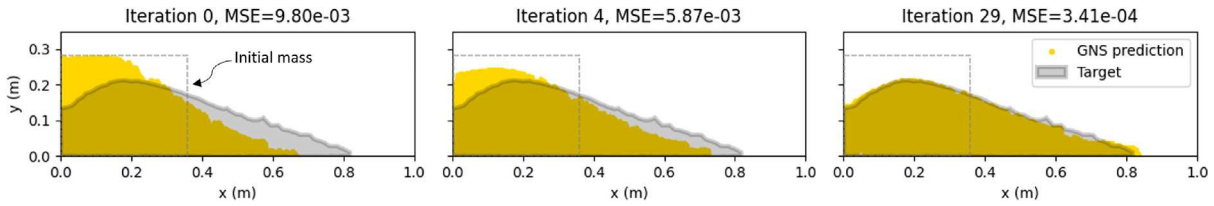

Fig. 10. Visualization of GNS prediction during optimization with different $\boldsymbol{v}$. The yellow dots represent the final deposit predicted by GNS ($\boldsymbol{S}_{\boldsymbol{v}}$) and the gray shade represents the target final deposit from MPM with $\boldsymbol{v}_{target}$ ($\boldsymbol{S}_{\boldsymbol{v}_{target}}$).

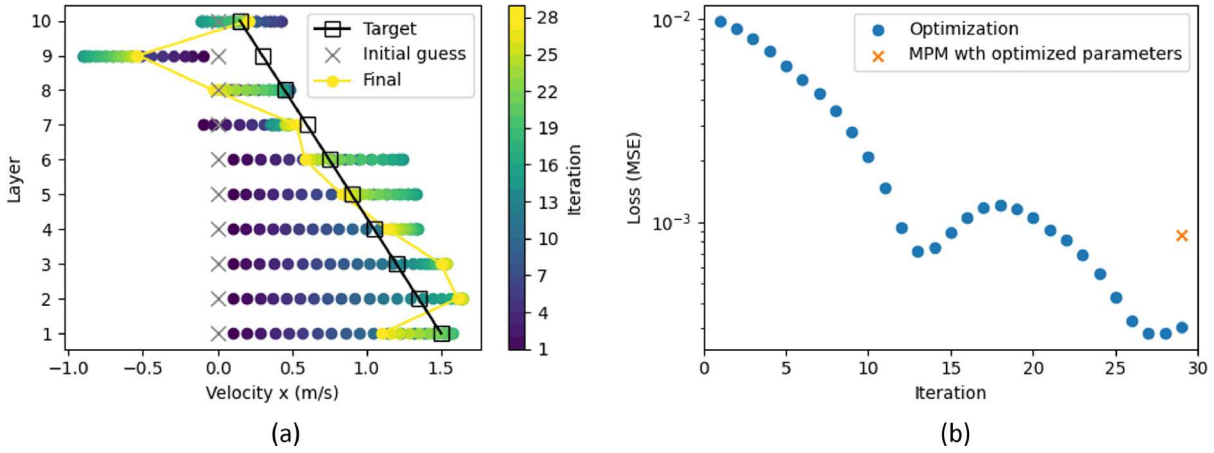

Fig. 11. Optimization history for the layer velocities of granular mass: (a) velocities for each layer, (b) loss.

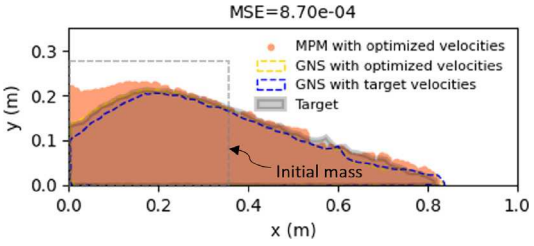

Fig. 12. MPM simulation results of the final deposit from the optimized velocities (red dots) and target velocities (gray shade), plotted together with the GNS prediction with target velocities (blue dashed line). The GNS prediction of the final deposit with optimized velocities is plotted with the yellow dashed line, which is the same as the deposit geometry from Fig. 10 at the 29th iteration. The MSE at the top of the figure is for the error between the MPM with optimized velocities and the target.

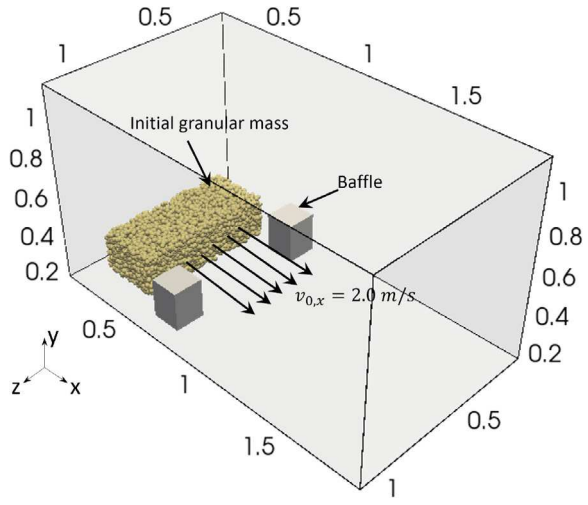

Fig. 13. Initial geometry of the granular mass.

$\phi + \Delta\phi$ for estimating the gradient using FD. The computation is conducted on 56 Intel Cascade Lake processor cores on the Texas Advanced Computing Center (TACC) Frontera. The mean computation time for the MPM forward simulation takes 8078 s with a standard deviation of 381 s, and it requires the same amount of time to conduct another forward simulation at $\phi + \Delta\phi$ to compute the gradient. Consequently, one optimization iteration requires approximately $8078 \times 2 = 16\,157$ s, which makes the ϕ estimation unreasonably time-consuming. Our approach (AD-GNS) does the forward simulation only at ϕ, and the gradient is computed based on reverse-mode AD. The computation is conducted on RTX with 16 GB memory on TACC Frontera. The forward simulation takes 33 s with the standard deviation of 2.01 s, and the gradient computation using backpropagation takes 74 s with the standard deviation of 1.36 s. Consequently, one optimization iteration requires approximately 107 s on average, outperforming our baseline case by 151 times speedup. The optimization result infers accurate ϕ with 6.06% of error as discussed in Table 2.

We also evaluate the computation time for FD with GNS (see Table 3). The gradient computation from FD with GNS requires less time than AD since it only requires one more forward evaluation in this single parameter case. However, the optimization result from FD shows the unreasonable ϕ inference due to inaccurate gradient estimation, in contrast to AD, which infers accurate ϕ.

For more complex inverse problems that involve high-dimensional parameter space, such as the cases in Sections 4.2 and 4.3, FD with MPM becomes almost infeasible due to the computation intensity. Although FD with GNS can return the gradient values faster than MPM, the computation time proportionally increases with the number of parameters, and the gradient values are inaccurate. In contrast, our approach (AD with GNS) can still accomplish efficient and accurate gradient computation owing to the reverse-mode AD despite the increase in the number of parameters, resulting in the successful parameter inference as we showed in Section 4.2 or Section 4.3.

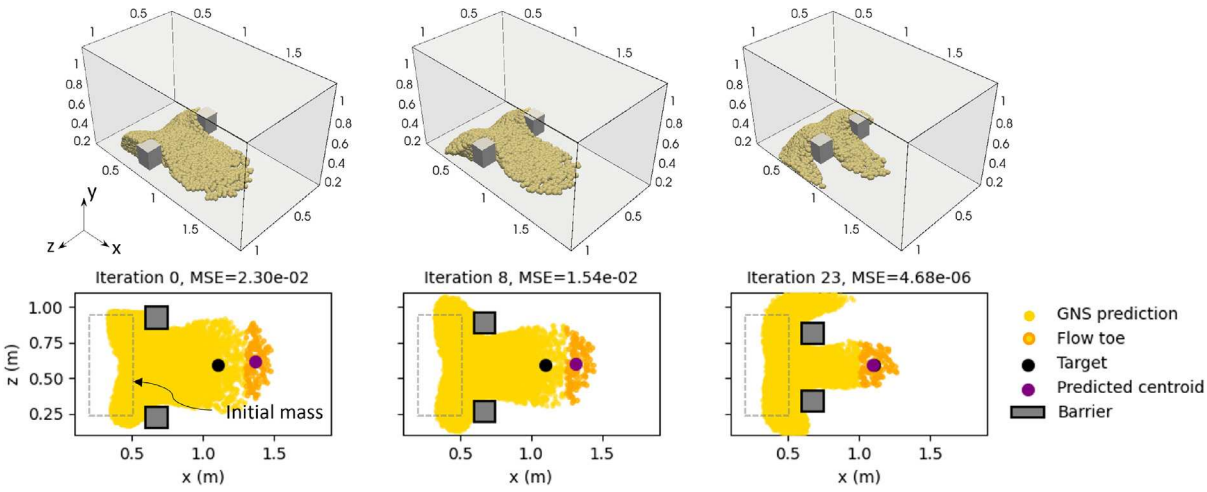

Fig. 14. Optimization history for baffle locations.

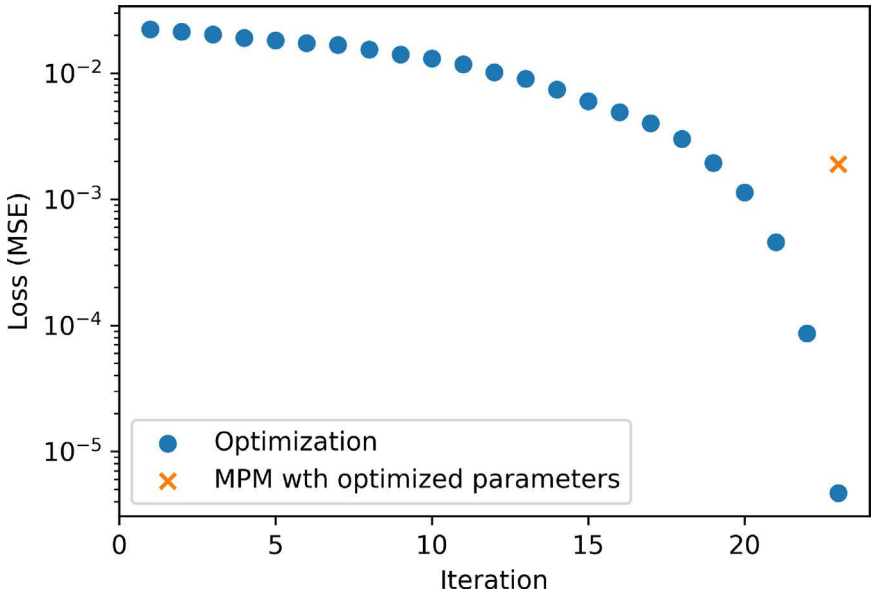

Fig. 15. Loss history for the centroid location.

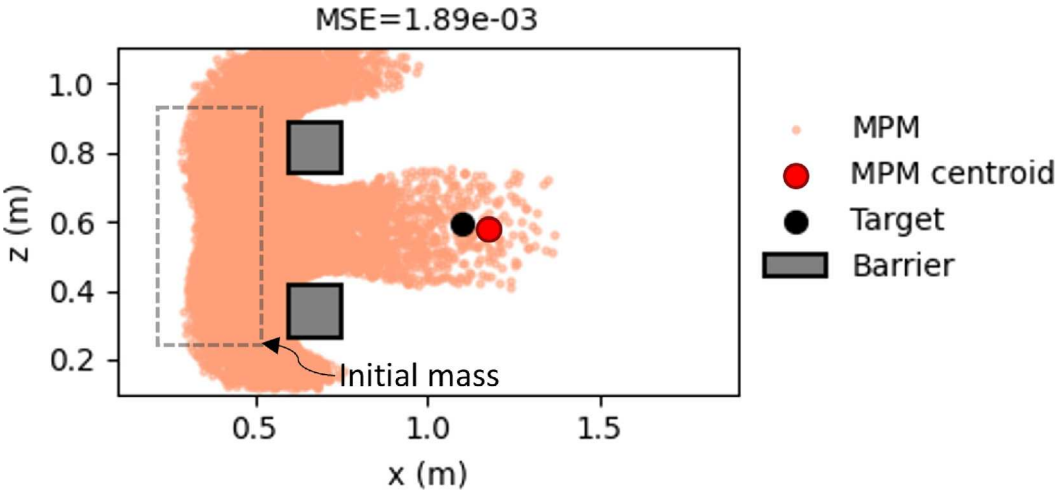

Fig. 16. MPM simulation results of final deposit from the inferred baffle locations and its flow toe centroid compared to the target.

Table 3
Mean computation times each optimization iteration for different gradient computation methods. Values are reported based on the short column ($a = 2.0$) with $\phi = 42°$ for 374 timesteps of simulation duration.

Method	Forward simulation (s)		Gradient computation (s)		Optimization iteration (s)	Estimated ϕ	ϕ error (%)
	Mean	std	Mean	std			
FD with MPM	8078	381	2x of forward		16 157	N/A	N/A
AD with GNS	33	2.01	74	1.36	107	44.54	6.06
FD with GNS	33	1.03	2x of forward		66	37.86	9.84

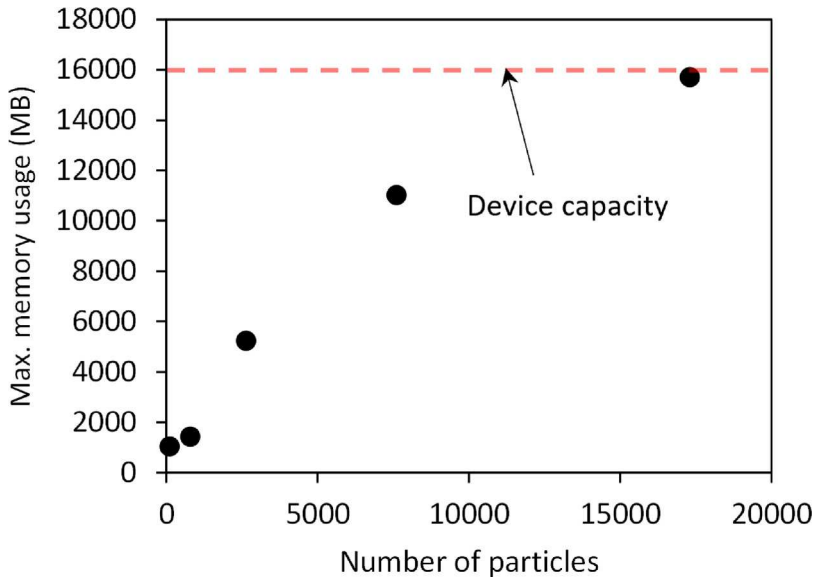

Fig. 17. Peak memory usage during backpropagation across up to 400 forward computation steps in relation to an increasing number of particles.

5. Limitations

GNS is an efficient surrogate model for forward simulation. However, generalization beyond training data may not perfectly replicate the high fidelity ground truth behaviors, resulting in some inevitable error, as shown in Figs. 12 and 16. To mitigate this, expanding the training dataset to encompass a more comprehensive array of scenarios could help reduce the prediction error.

While gradient checkpointing offers a workaround for the high memory demands associated with reverse-mode AD in GNN, tracking the gradient over the GNN with large graphs is still expensive. We evaluated peak memory usage during backpropagation across up to 400 forward computation steps with increasing particles, as depicted in Fig. 17. At around 20K particles, the backpropagation process exceeds the memory capacity of our device (16 GB, RTX node at Frontera, TACC), failing. Introducing checkpoints more frequently can reduce memory consumption; however, this strategy requires additional computations for the forward pass, leading to a trade-off between memory efficiency and computational overhead. Another strategy to alleviate memory constraints involves partitioning the graph (Çatalyürek et al., 2023) and employing multi-GPU processing. Our GNS implementation (Kumar and Vantassel, 2022) supports efficient multi-GPU processing. However, effectively partitioning large graphs and facilitating communication across these partitions in a multi-GPU environment presents challenges and is the subject of active research.

6. Conclusion

This study introduces an AD-based efficient framework for solving complex inverse problems in granular flows using graph neural network-based simulators (GNS). By leveraging the computational efficiency, differentiability, and generalization capabilities of GNS, coupled with gradient-based optimization through reverse-mode AD, our approach successfully identifies optimal parameters to achieve desired outcomes in diverse granular flow scenarios.

We demonstrate the effectiveness of our methodology in solving single-parameter and multi-parameter inverse analysis and design problems in granular flows. The single parameter optimization achieves <8.94% error in estimating ϕ compared to ϕ_{target}. In the multi-parameter case, the initial velocity is inferred within 8.70e−4 MSE. For the design problem, our method identifies a reasonable baffle arrangement with 1.89e−3 MSE, which is about an order of magnitude improved than the initial guess (2.30e−2). Despite the test configurations being outside the training distribution, our technique generalizes well, highlighting the flexibility of the methodology. Inverse analysis with AD-GNS shows reasonable parameter estimations over a wide range of inverse problems. Nevertheless, the surrogate nature of GNS means some errors are inevitable compared to a high-fidelity simulator. However, the result from our method provides useful information on parameters for further analyses using high-fidelity solvers.

The proposed AD-GNS framework solves inverse problems efficiently on a single GPU, achieving 151x speedups compared to forward high-fidelity simulators with finite difference gradients. Furthermore, the integration of gradient checkpointing enables scaling to complex 3D dynamics over hundreds of timesteps that would otherwise be infeasible due to the excessive memory consumption for backpropagation.

Overall, this study highlights the prospect of data-driven differentiable surrogates in inverse modeling of granular flow hazards.

Data and code availability

The code for GNS (Kumar and Vantassel, 2022) and inverse problems are available under the MIT license on GitHub (https://github.com/geoelements/gns and https://github.com/geoelements/gns-invers

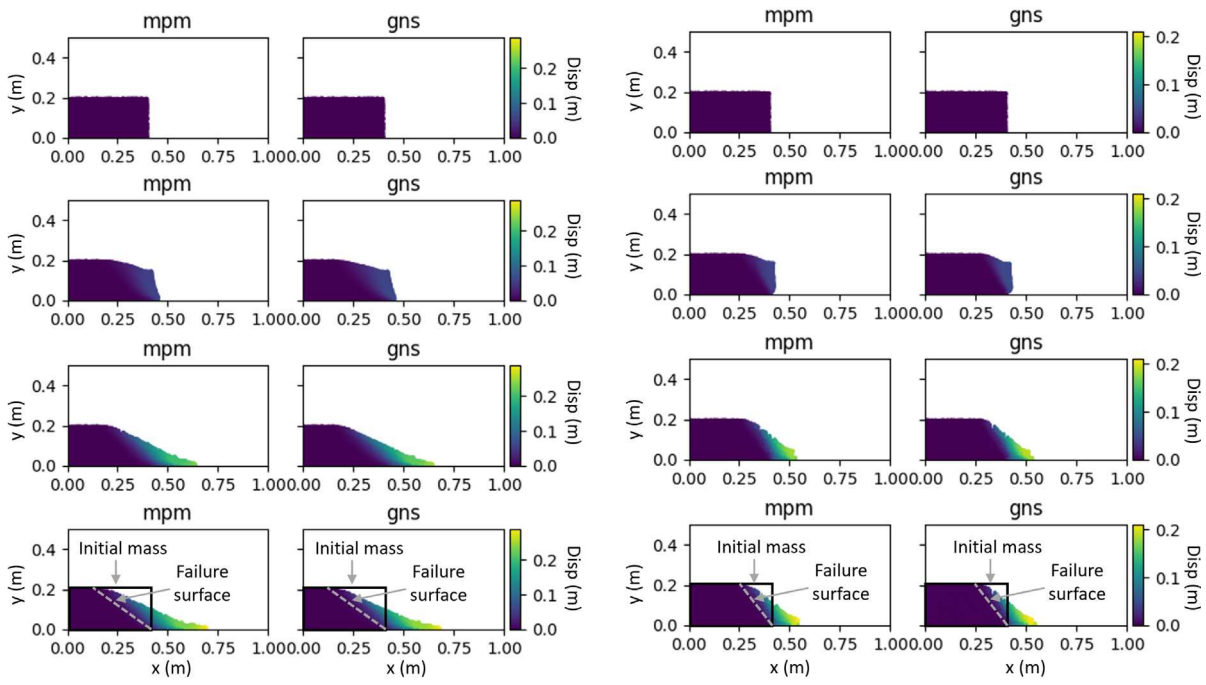

Fig. A.1. Evolution of material point flow with normalized time for GNS and MPM for the short column with a = 0.5: (a) $\phi = 21°$, (b) $\phi = 42°$. The color represents the magnitude of the displacement. Each row corresponds to a column before the flow initiation, $t/\tau_c = 1.0$, $t/\tau_c = 2.5$, and the final deposit at the last timestep.

e-examples). The training dataset, trained models, and the data used for the inverse analysis are published under CC-By license on DesignSafe Data Depot (Choi and Kumar, 2023b, 2024).

CRediT authorship contribution statement

Yongjin Choi: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Formal analysis, Data curation. **Krishna Kumar:** Writing – review & editing, Supervision, Software, 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

Data will be made available on request.

Acknowledgments

This material is based upon work supported by the National Science Foundation, United States under Grant No. #2103937 and #2229702. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing Frontera and Lonestar6 HPC resources to support GNS training (https://www.tacc.utexas.edu).

Appendix A. Performance of GNS

As a benchmark test for Flow2D, we show the performance of GNS using the granular column collapse. Granular column collapse (Lajeunesse et al., 2005; Utili et al., 2015; Lube et al., 2005; Kumar, 2015) is an experiment to study the dynamics of granular flows in a controlled setting. A granular column of initial height H_0 and length L_0 is placed on a flat surface and allowed to collapse under gravity. The dynamics of granular column collapse is majorly governed by the initial aspect ratio of the column ($a = H_0/L_0$). The previous study (Choi and Kumar, 2023a) proves that GNS can accurately predict different flow dynamics of granular columns with various aspect ratios beyond which the data the model is trained. In this paper, we briefly show the prediction performance of the GNS used in this study on a short column ($a = 0.5$) with different friction angles ($\phi = 21°$ and $42°$), whose aspect ratio and friction angles are not seen during the training.

Fig. A.1 shows the evolution of granular flow for the short column ($a = 0.5$) with normalized time (t/τ_c) simulated by GNS and MPM. Here, we assume the MPM result is the ground truth. t is physical time, and τ_c is the critical time, defined as the time required for the flow to mobilize fully. τ_c is defined as $\sqrt{H_0/g}$, where g is the gravitational acceleration. Figs. A.1(a) and A.1(b) is the result with $\phi = 21°$ and $\phi = 42°$. Each row of the figure corresponds to flow (1) at the time before initiation of collapse, (2) $t/\tau_c = 1.0$, (3) 2.5, and (4) at the last timestep ($k = 380$) when the flow reaches static equilibrium and forms the final deposit.

Generally, the collapse shows three stages for both friction angles. First, the flow is mobilized by the failure of the flank and reaches full mobilization around $t/\tau_c = 1.0$. Next, the majority of the runout occurs until $t/\tau_c = 2.5$. Beyond $t/\tau_c = 2.5$, the spreading decelerates due to the basal friction and finally stops. Fig. A.1 shows GNS successfully captures this overall flow progress stages for the different friction angles.

Although the overall progress of the flow is similar, the detailed behaviors diverge depending on the friction angles. When ϕ is small (Fig. A.1a), a larger amount of flow mass is mobilized along the flank of the column above the failure surface until $t/\tau_c = 1.0$ compared to

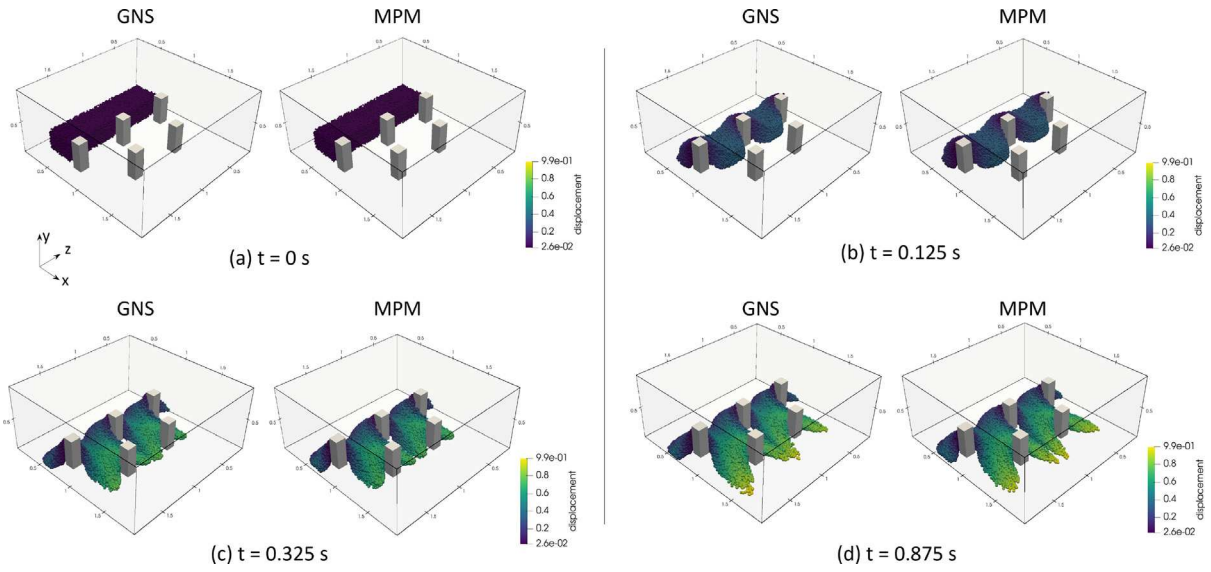

Fig. A.2. Evolution of flow interacting with baffles for GNS and MPM from initial condition to the final deposit. The simulation domain is 1.8 × 0.8 × 1.8 m, and the initial geometry of the granular mass is 0.35 × 0.25 × 1.4 m. The barrier size is 0.15 × 0.30 × 0.15 m. The center location of the three baffles in the first row is at ($x = 0.76$, $z = 0.36$) m, ($x = 0.76$, $z = 1.00$) m, and ($x = 0.76$, $z = 1.64$) m, and the two baffles in the second row are at ($x = 1.26$, $z = 0.68$) m and ($x = 1.26$, $z = 1.32$) m.

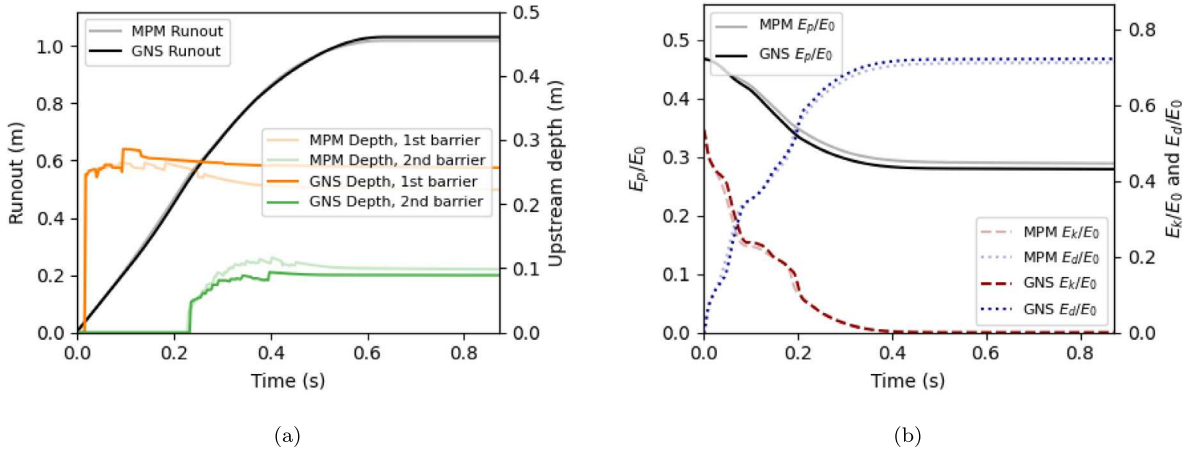

Fig. A.3. Quantitative comparison between MPM and GNS: (a) Runout and upstream depth evolution with time, (b) potential, kinetic and dissipation energy (E_p, E_k, and E_d) evolution with time normalized by E_0, where E_0 is total energy at $t = 0$.

the column with larger ϕ (Fig. A.1b). The greater mobilization of the flank failure in small ϕ leads to a longer runout until the end of the flow than that of large ϕ. At the end of the flow, a greater amount of static soil mass is observed below the failure surface for larger ϕ with a truncated conical shape. In addition, a smaller plateau is observed at the top of the final deposit in the case of smaller ϕ. The visual comparison of the flow profile (Fig. A.1) shows that GNS well replicates the different granular flow behaviors depending on friction angle with the final runout error of 2.09% and 1.17% for each case.

Fig. A.2 shows an example of GNS prediction trained on Obstacle3D datasets. The test configuration is outside the training data distribution as summarized in 1. Specifically, the simulation domain is four times larger, with 2.2 times more material points than the training data. The maximum length of the initial granular mass in our test is 1.7 m, while that of training data is 0.7 m. Additionally, we test the GNS with five baffles, versus one to three baffles in the training data.

From the initial state to t = 0.125 s (Fig. A.2a–b), the granular debris propagates downstream uniformly and impacts the first baffle row. Upon hitting this row, the baffles obstruct the sides of the flow while the material between the baffles proceeds towards the next row. Concurrently, material deposits and builds up upstream of the baffles, reaching almost the height of the baffle. From t = 0.125 s to t = 0.325 s (Fig. A.2b–c), the flow impacts the next baffle row and diverges into four granular jets through the open area with roughly symmetric shapes. Less material deposition forms upstream of the last baffle row than the first. Beyond t = 0.325 s (Fig. A.2c–d), the spreading of the grains decelerates due to basal friction and finally reaches static equilibrium around t = 0.875 s. At this stage, the runout error between GNS and MPM is 0.87%. The GNS rollout successfully replicates the overall kinematics, including complex baffle interactions.

To quantitatively compare the results from the GNS and MPM, the evolution of the runout distance is measured (Fig. A.3a). Here, runout is defined as the flow's maximum travel distance compared to the initial granular mass geometry. As flow initiates, the runout rapidly increases until 0.325. Subsequently, the runout gradually decelerates and eventually stops due to the interaction with the baffles and frictional dissipation. GNS accurately predicts the runout evolution simulated by MPM despite the simulation including more barriers and a larger domain than the training data.

The upstream depth evolution of the granular mass deposition behind the baffles is also measured (Fig. A.3a). The upstream flow depth refers to the depth measured upstream of the baffle array when

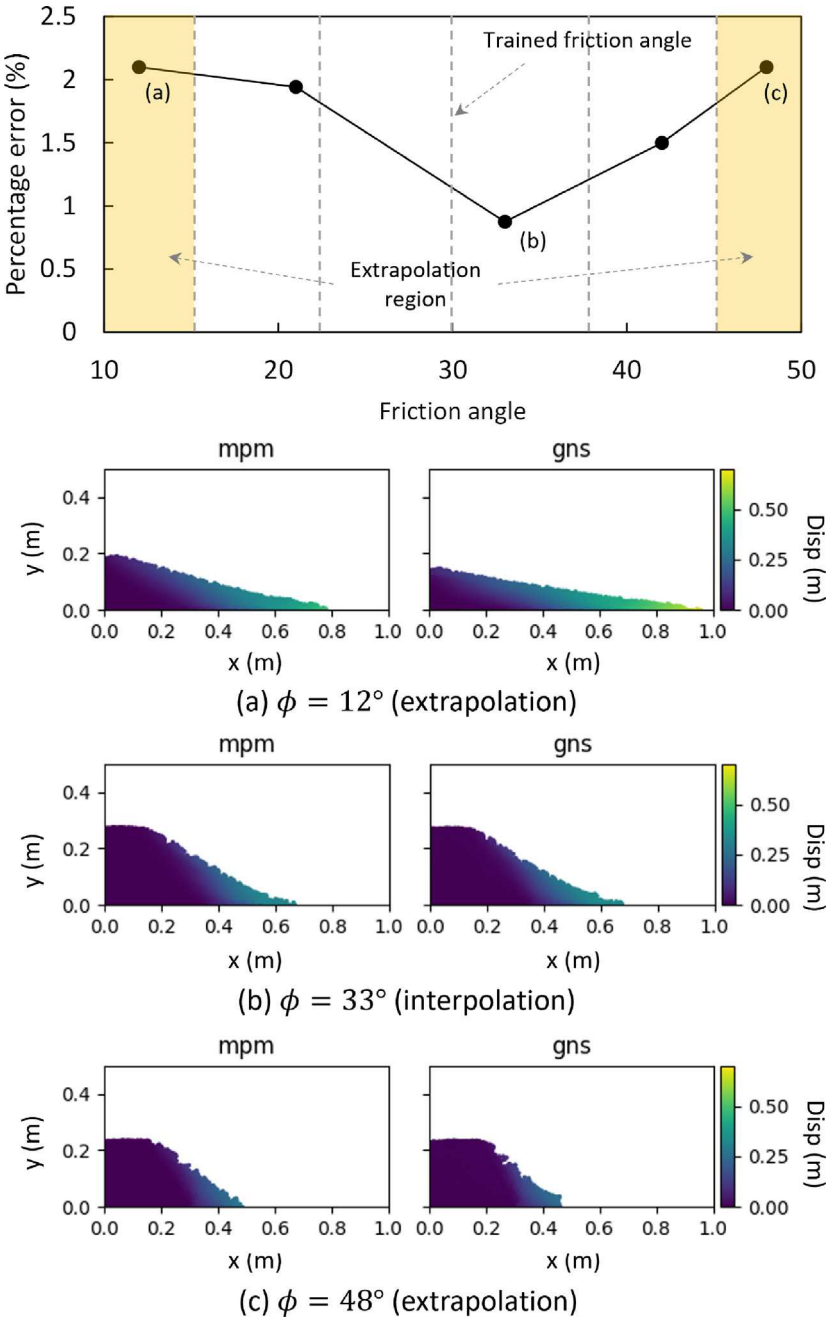

Fig. A.4. Normalized runout error between MPM and GNS evaluated at different friction angles ϕ. The final deposit at points (a), (b), and (c) is shown below in the plot for the normalized runout error.

the flow impacts the baffles. As the runout proceeds, the flow faces the first and then a second row of baffles. Upon impact with the first baffles, the upstream depth spikes close to the baffle height (0.3 m) due to the deposition of the granular mass. It then decreases slightly over time as the flow around the baffles reaches equilibrium. When the flow hits the next baffle row, the upstream depth surges again and gradually decreases. GNS accurately replicates the overall upstream depth change trend with minor errors.

Fig. A.3b shows the energy evolution. The initial granular flow shows high kinetic and potential energy. Subsequent granular flow and its blockage due to the baffles cause the kinetic and potential energy drop. GNS precisely captures these energy evolution trends. The results from Fig. A.3 suggest that GNS successfully learns the granular flow dynamics with obstacle interactions.

As explained in Table 1, the dataset for training the 2D GNS (Flow2D) only contains the granular flow trajectories with the friction angles $\phi = 15,\ 22.5,\ 30,\ 37.5,\ 45°$. Here, we show the performance of GNS on the friction angles not observed during training. The Fig. A.4 shows the normalized runout prediction error (%) for the granular column with $a = 0.8$. The gray dashed lines represent the training ϕ. The test is conducted with $\phi = 12,\ 21,\ 33,\ 42,\ 48°$. In particular, $\phi = 12°$ and $48°$ lies on the extrapolation region as highlighted in Fig. A.4. Although the extrapolation shows a slightly higher error than the predictions on the interpolation region ($\phi = 21,\ 33,\ 42°$), the GNS

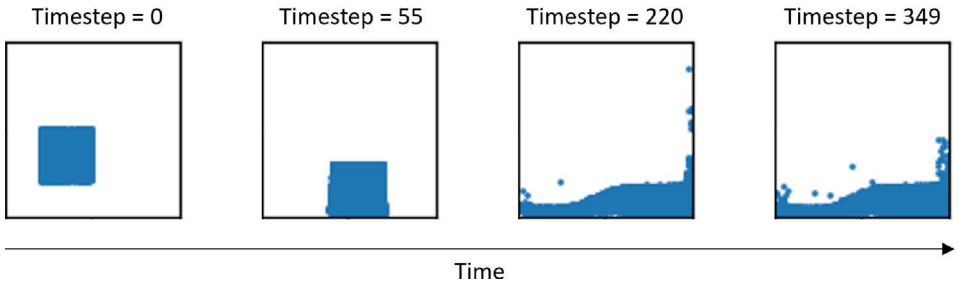

Fig. B.1. An example of training data for 2D GNS (Flow2D).

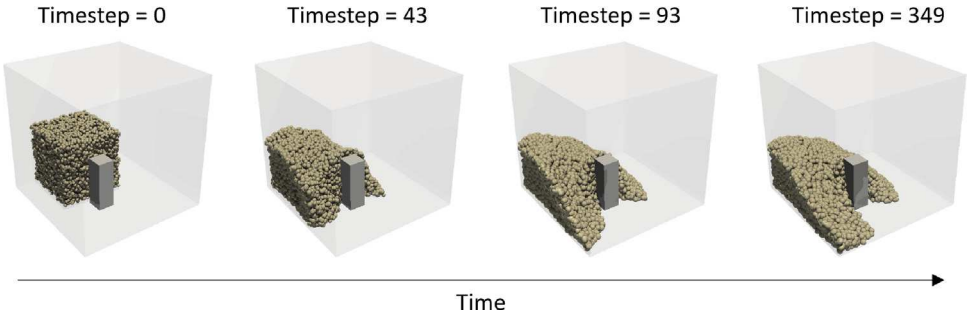

Fig. B.2. An example of training data for 3D GNS (Obstacle3D).

produces good runout predictions for the friction angles not seen during the training.

Appendix B. Training data

See Figs. B.1 and B.2.

References

Abraham, M.T., Satyam, N., Reddy, S.K.P., Pradhan, B., 2021. Runout modeling and calibration of friction parameters of Kurichermala debris flow, India. Landslides 18 (2), 737–754. http://dx.doi.org/10.1007/s10346-020-01540-1.

Allen, K.R., Lopez-Guevara, T., Stachenfeld, K., Sanchez-Gonzalez, A., Battaglia, P., Hamrick, J., Pfaff, T., 2022. Physical design using differentiable learned simulators. arXiv preprint arXiv:2202.00728.

Babu, G.S., Basha, B.M., 2008. Optimum design of cantilever sheet pile walls in sandy soils using inverse reliability approach. Comput. Geotech. 35 (2), 134–143.

Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V.F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H.F., Ballard, A.J., Gilmer, J., Dahl, G.E., Vaswani, A., Allen, K.R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M.M., Vinyals, O., Li, Y., Pascanu, R., 2018. Relational inductive biases, deep learning, and graph networks. CoRR abs/1806.01261.

Battaglia, P.W., Pascanu, R., Lai, M., Rezende, D.J., Kavukcuoglu, K., 2016. Interaction networks for learning about objects, relations and physics. CoRR abs/1612.00222.

Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M., 2018. Automatic differentiation in machine learning: a survey. J. Marchine Learn. Res. 18, 1–43.

Calvello, M., Cuomo, S., Ghasemi, P., 2017. The role of observations in the inverse analysis of landslide propagation. Comput. Geotech. 92, 11–21.

Çatalyürek, Ü., Devine, K., Faraj, M., Gottesbüren, L., Heuer, T., Meyerhenke, H., Sanders, P., Schlag, S., Schulz, C., Seemaier, D., et al., 2023. More recent advances in (hyper) graph partitioning. ACM Comput. Surv. 55 (12), 1–38.

Chen, T., Xu, B., Zhang, C., Guestrin, C., 2016. Training deep nets with sublinear memory cost. CoRR abs/1604.06174 arXiv:1604.06174 URL http://arxiv.org/abs/1604.06174.

Cheylan, I., Fritz, G., Ricot, D., Sagaut, P., 2019. Shape optimization using the adjoint lattice Boltzmann method for aerodynamic applications. AIAA J. 57 (7), 2758–2773.

Choi, Y., Kumar, K., 2023a. Graph Neural Network-based surrogate model for granular flows. arXiv:00.

Choi, Y., Kumar, K., 2023b. Training, validation, testing data, and trained model. http://dx.doi.org/10.17603/DS2-4NQZ-S548, URL https://www.designsafe-ci.org/data/browser/public/designsafe.storage.published/PRJ-4275/#details-5385005666722770450-242ac117-0001-012.

Choi, Y., Kumar, K., 2024. Solving inverse problems using differentiable graph neural network simulator. http://dx.doi.org/10.17603/DS2-0WJQ-0J84, URL https://www.designsafe-ci.org/data/browser/public/designsafe.storage.published/PRJ-4534.

Christen, M., Kowalski, J., Bartelt, P., 2010. RAMMS: Numerical simulation of dense snow avalanches in three-dimensional terrain. Cold Reg. Sci. & Technol. 63 (1), 1–14. http://dx.doi.org/10.1016/j.coldregions.2010.04.005, URL https://www.sciencedirect.com/science/article/pii/S0165232X10000844.

Cuomo, S., Calvello, M., Villari, V., 2015. Inverse analysis for rheology calibration in SPH analysis of landslide run-out. In: Engineering Geology for Society and Territory-Volume 2: Landslide Processes. Springer, pp. 1635–1639.

Dhara, A., Sen, M.K., 2023. Elastic full waveform inversion using a physics guided deep convolutional encoder-decoder. IEEE Trans. Geosci. Remote Sens..

Ensor, K.B., Glynn, P.W., 1997. Stochastic optimization via grid search. Lect. Appl. Math.-Am. Math. Soc. 33, 89–100.

Frazier, P.I., 2018. A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811.

Hecht-Nielsen, R., 1992. Theory of the backpropagation neural network. In: Neural Networks for Perception. Elsevier, pp. 65–93.

Ho, S.L., Yang, S., 2010. The cross-entropy method and its application to inverse problems. IEEE Trans. Magn. 46 (8), 3401–3404.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., Jiang, C., 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Trans. Graph. 37 (4), 150.

Hu, Y., Liu, J., Spielberg, A., Tenenbaum, J.B., Freeman, W.T., Wu, J., Rus, D., Matusik, W., 2019. Chainqueen: A real-time differentiable physical simulator for soft robotics. In: 2019 International Conference on Robotics and Automation. ICRA, IEEE, pp. 6265–6271.

Hungr, O., McDougall, S., 2009. Two numerical models for landslide dynamic analysis. Comput. Geosci. 35 (5), 978–992.

Ju, L.-Y., Xiao, T., He, J., Wang, H.-J., Zhang, L.-M., 2022. Predicting landslide runout paths using terrain matching-targeted machine learning. Eng. Geol. 311, http://dx.doi.org/10.1016/j.enggeo.2022.106902.

Kermani, E., Qiu, T., Li, T., 2015. Simulation of collapse of granular columns using the discrete element method. Int. J. Geomech. 15 (6), 04015004. http://dx.doi.org/10.1061/(asce)gm.1943-5622.0000467.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kumar, K., 2015. Multi-Scale Multiphase Modelling of Granular Flows (Ph.D. thesis, Ph.D. thesis). University of Cambridge, http://dx.doi.org/10.5281/zenodo.160339.

Kumar, K., Delenne, J.Y., Soga, K., 2017a. Mechanics of granular column collapse in fluid at varying slope angles. J. Hydrodyn. 29 (4), 529–541. http://dx.doi.org/10.1016/s1001-6058(16)60766-7.

Kumar, K., Salmond, J., Kularathna, S., Wilkes, C., Tjung, E., Biscontin, G., Soga, K., 2019. Scalable and modular material point method for large-scale simulations. arXiv:1909.13380.

Kumar, K., Soga, K., Delenne, J.-Y., Radjai, F., 2017b. Modelling transient dynamics of granular slopes: MPM and DEM. Procedia Eng. 175, 94–101. http://dx.doi.org/10.1016/j.proeng.2017.01.032.

Kumar, K., Vantassel, J., 2022. GNS: A generalizable graph neural network-based simulator for particulate and fluid modeling. arXiv:2211.10228.

Lajeunesse, E., Monnier, J.B., Homsy, G.M., 2005. Granular slumping on a horizontal surface. Phys. Fluids 17 (10), 103302. http://dx.doi.org/10.1063/1.2087687.

LeCun, Y., Touresky, D., Hinton, G., Sejnowski, T., 1988. A theoretical framework for back-propagation. In: Proceedings of the 1988 Connectionist Models Summer School. Vol. 1, San Mateo, CA, USA, pp. 21–28.

Liu, X., Liu, Y., Li, X., Yang, Z., Jiang, S.-H., 2023. Efficient adaptive reliability-based design optimization for geotechnical structures with multiple design parameters. Comput. Geotech. 162, 105675.

Lube, G., Huppert, H.E., Sparks, R.S.J., Freundt, A., 2005. Collapses of two-dimensional granular columns. Phys. Rev. E 72 (4), http://dx.doi.org/10.1103/physreve.72.041301.

Mast, C.M., Arduino, P., Mackenzie-Helnwein, P., Miller, G.R., 2014. Simulating granular column collapse using the Material Point Method. Acta Geotech. 10 (1), 101–116. http://dx.doi.org/10.1007/s11440-014-0309-0.

Mergili, M., Fischer, J.-T., Krenn, J., Pudasaini, S.P., 2017. R.avaflow v1, an advanced open-source computational framework for the propagation and interaction of two-phase mass flows. Geosci. Model Dev. 10 (2), 553–569. http://dx.doi.org/10.5194/gmd-10-553-2017, URL https://gmd.copernicus.org/articles/10/553/2017/.

Pires, C., Miranda, P.M., 2001. Tsunami waveform inversion by adjoint methods. J. Geophys. Res.: Oceans 106 (C9), 19773–19796.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., Battaglia, P.W., 2020. Learning to simulate complex physics with graph networks. CoRR abs/2002.09405 arXiv:2002.09405 URL https://arxiv.org/abs/2002.09405.

Soga, K., Alonso, E., Yerro, A., Kumar, K., Bandara, S., 2016. Trends in large-deformation analysis of landslide mass movements with particular emphasis on the material point method. Géotechnique 66 (3), 248–273. http://dx.doi.org/10.1680/jgeot.15.LM.005.

Staron, L., Hinch, E.J., 2005. Study of the collapse of granular columns using two-dimensional discrete-grain simulation. J. Fluid Mech. 545, 1–27. http://dx.doi.org/10.1017/S0022112005006415.

Utili, S., Zhao, T., Houlsby, G.T., 2015. 3D DEM investigation of granular column collapse: Evaluation of debris motion and its destructive power. Eng. Geol. 186, 3–16. http://dx.doi.org/10.1016/j.enggeo.2014.08.018.

Wang, Q., Kumar, K., 2023. An inverse analysis of fluid flow through granular media using differentiable lattice Boltzmann method. arXiv preprint arXiv:2310.00810.

Yang, Q., Hambleton, J.P., 2021. Data-driven modeling of granular column collapse. In: Geo-Extreme 2021. pp. 79–88. http://dx.doi.org/10.1061/9780784483701.008.

Zeng, P., Sun, X., Xu, Q., Li, T., Zhang, T., 2021. 3D probabilistic landslide run-out hazard evaluation for quantitative risk assessment purposes. Eng. Geol. 293, 106303. http://dx.doi.org/10.1016/j.enggeo.2021.106303.

Zhao, Q., Lindell, D.B., Wetzstein, G., 2022. Learning to solve pde-constrained inverse problems with graph networks. arXiv preprint arXiv:2206.00711.